An Image Classification Tool of Wikimedia Commons

Master's Thesis submitted

to

Advisor

Dr. Sigbert Klinke

Examiners

Prof. Dr. Sonja Greven

Prof. Dr. Weining Wang

Humboldt-Universität zu Berlin

School of Business and Economics

Chair of Statistics

by

Sisi Huang

(526628)

in partial fulfillment of the requirements

for the degree of

Master of Science

Berlin, June 10, 2020

Acknowledgement

I would like to thank Dr. Klinke, who inspired the idea of this thesis, and provided me with extraordinary help. Furthermore, I wish to show my gratitude to Professor Greven and professor Wang for their guidance and feedback in the whole process. And also thanks for the team in Humboldt Lab for Empirical and Quantitative Research, without their technical support, this project would have been impossible. Last but not least, the biggest appreciation to my friends Guangyu He, Thomas Siskos and Lin Yang, and my family for all the support throughout this thesis, they gave me strength and courage to overcome difficulties during the research.

Abstract

Labelling massive datasets consisting of images from webpages manually is quite time-consuming and also exhausting. If there was a tool which can help us to classify those unlabeled images automatically, it would not overwhelm us nearly as much. In this thesis we aim to extract significant features from images and to automate the annotation of unlabeled images.

Due to the variety of images, we focus our attention on solving the problem of chart image classification. Chart images are frequently presented in documents and used as a common tool for visualizing relationships within the data. Especially, they are able to distinguish themselves by their patterns or shapes. To deal with this problem we propose machine learning models that can extract the images' features automatically, and predict their labels. Convolutional neural networks are the popular models for solving such problem of image classification. Thus, it is our goal to bridge the relationship between chart images and neural networks.

In this thesis we attempt two directions to implement convolutional neural networks: transfer learning and self-training models. On a set of testing data a model using transfer learning based on the VGG-16 pre-trained model, achieves a test accuracy of up to 0.65. Self-training models are LeNet-5, Alex blocks and VGG blocks, which are grounded by AlexNet and VGG. However, performances of self-training models are sightly worse than transfer learning, the highest prediction accuracy of the self-training models is only 0.47.

Key words: Image Labelling, Convolutional Neural Network, Transfer learning, Machine Learning

Contents

List of Abbreviations

USPTO	United States Patent and Trademark Office
CNN	Convolutional neural network
CFD	Cumulative frequency distribution
FP	Frequency Polygon
SMOTE	Synthetic Minority Over-sampling Technique Learning
KNN	K-Nearest Neighbors
RMSE	Root Mean Square Error
SVM	Support Vector machine
GPU	Graphics Processing unit
ReLu	Rectifier Linear unit
VGG	Visual Geometry Group
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

List of Figures

List of Tables

1 Introduction

Image classification aims to extract features and insights of images from different classes and understand the structure of images, then sort unclassified images into their corresponding class. In recent years the amount of images has exploded to a ridiculous degree in the real world. It is not an easy task to classify those massive images by hand. Therefore, how to conduct the machine to extract and interpret the insights of images automatically will be a big challenge.

Of course, the research on image classification did not happen overnight. In 1963, Roberts [22] came forth with the idea, that the visual world can be simplified into simple geometric shapes and the goal is to reconstruct the shapes of photographs. Marr [18] purposed a theory: stage of vision, that is processing the two-dimensional array from original input to three-dimensional model, which provided a new insight to improve the performance of computer recognition in the visual world. Later, Fukushima [8] proposed one of the first neural network model's pattern: Neocognitron, which made the network self-learning possible. Y.LeCun [32] , who was inspired by Fukushima's system, developed a hierarchical neural network called LeNet, and it has been successfully applied to the handwritten character recognition. Krizhevsky et al. [15] formulated AlexNet, which achieved the extraordinary performance in an image classification competition: ImageNet Large Scale Visual Recognition Challenge. Its architecture was quit similar with LeNet, but with more and deeper layers in the neural network. Furthermore, AlexNet also promoted the utilization of graphics processing unit (GPU) for speeding up the training process. Since then, neural network is becoming more and more popular for image classification tasks.

1.1 Background of Wikimedia Commons

Wikimedia Commons is a part of the free-to-use Wikipedia family. More precisely, it is an online repository of media documents like images and videos. Erik Möller first proposed this project in March 2004, later it was launched by Wikimedia Foundation in September 7, 2004. The initial inclination was to reduce the duplication of media files in all Wikipedia projects, which can productively avoid redundant documents in different wikis. As in other Wikipedia projects, in Wikimedia Commons, users can also upload, edit and download any media files they want and enrich the platform with all kinds of gratis accessible media files, those files are also shared for every Wiki-project. Until October, 2018 there were already up to 50 million media files, of which more than 10% were images. Now it even contains over 55 million media

files, which are managed and edited by registered volunteers.

Actually, in Wikimedia Commons, users already set up their own classification system. The classification system based on the criterion of the United States Patent and Trademark Office (USPTO), which has been developed for trademark registration for years. Hence, classifications are on the basis of the content of images, coded by a six-digit number, and images would be manually collected in the category. Furthermore, most of images are stored in their corresponding category, even corresponding subcategory or sub-subcategory, then users can easily detect the category or subcategory by its digit number, and find images from different categories. However, due to an excess of information, more and more images are just uploaded into the system, but they are not put into right position yet. Therefore, labeling millions images by hand becomes a Sisyphean task.

1.2 Purpose

As it is mentioned in the last paragraph, Wikimedia Commons is an open source platform with over 5 million images, which means it is very time consuming for users to manually classify images that are out of classification. If there was a classification tool to help users to sort all unlabeled media objects into their own classes, it would accelerate the process of image classification in this platform, and avoid complex and tedious image recognition and image labelling as well. Furthermore, it would provide such convenience for users to search images they are looking for. However, it is challenging to build a tool for all types of image files, therefore in this thesis we will set our sights on classifying chart images.

Why chart images? Chart images are commonly used to illustrate data structures and the relationship among data in the digit document, which are not only applied in the academic field but also other areas. For example, line charts are broadly used in the marketing for recording the stock price tendency, pie charts are frequently applied for presenting the proportion of groups in the population. With the further development of the storage device, more and more chart images are able to be stored permanently. Furthermore, the demand of decoding hidden information from data also inspires the evolution of data visualization, of which chart image is such a powerful tool. Therefore, efficient chart recognition and chart classification [2, 13] are becoming essential. Besides that, chart images are created by different meanings and their shapes and patterns are variant and in general mutually exclusive. Analyzing chart images can be a good start for extracting the pattern inside images and identifying types by their own contents. Hence, this thesis will be specifically concerned with

chart images recognition and labeling unclassified chart images.

1.3 Thesis Structure

The following section 2 explains theoretical concepts involved with this work, in particular, we will show some important concepts of machine learning, image labeling and convolutional neural networks. Section 3 gives an insight of data preparation as well as image preprocessing. Section 4 introduces transfer learning and describes CNN models, as well as how and why an architecture of the CNN model is chosen. Section 5 discusses respective results with different CNN architectures. Conclusions and outlook for the further research will be handed out in the section 6.

2 Basic Concept

2.1 Labeling Image

Labeling images is a type of image classification task, which bases on extracting information and patterns of current classified images , applying that information to predict unclassified images and classifying into their corresponding classes. The idea of labeling image is to train the known-label image data, i.e. images in the category, test the unknown-label image data and classify unknown data into known class, which is very typical problem in supervised machine learning. Here are definitions of supervised learning and unsupervised learning.

- Supervised learning: is to learn the feature of a dataset, connecting the input and output with this feature, and each input example is also associated with a label or target [10]. More precisely, a supervised learning algorithm's task is to find out the mapping function from input to output by analyzing a known input dataset and its corresponding output dataset. When there is a new input data, this mapping function can predict an output for this new data. Supervised Learning makes use of solving two kinds of problems: regression and classification. Regression is typically used for quantitative output data, where it should find a linear or non-linear function to present the relationship among the data, for example, household expenditure and income. In this example, the expenditure is the input, and the income is the output, in general, more income in a family may cost more on household bills. On the contrary, the goal of classification is to recognize a mapping function, and predict a qualitative result by this mapping function, when the output data is a single category or multiple categories. For

example, cancer detection, which focuses on predicting whether the patient has cancer or not, and its output will only be yes or no, as such it is a binary classification task.

- Unsupervised learning: is to dig up features from a dataset, and learn useful properties of the structure of this dataset, but input data is without a label or target compared to supervised learning [10]. In unsupervised learning, the output data is unknown and unlabeled, the goal of the algorithm is to extract the covered structure or distribution of input data and to implement more about this data. For example, any clustering method is a good implementation for unsupervised learning, which is used to detect the similarity of data, gather the similar data together and explain the reasonability of each group.

Our dataset consists of various chart images, our target is to label unknown chart images from Wikimedia Commons by extracting features of labeled chart images, which is exactly a classic classification problem. Handling an image classification problem can encompass different approaches:

- K-Nearest neighbors (KNN), as a simple classification method, based on capturing the similarity of images, i.e. calculating the distance metrics among data, e.g. Euclidean distance suffices for the continuos data, but when dealing with the discrete data, the Hamming distance is a better approach. After that, ordering the data by distance, finding a optimal K based on Root Mean Square Error (RMSE) is the way to implement this algorithm. This is a simple, lazy learning and effortless algorithm for implementation. However, it's also strictly limited by increasing the volume of data, or training with high-dimensional data like images, especially, distances over high-dimensional data are very implausible. N. Kouiroukidis [20] has pointed out that the distance between nearest point and farthest point in the high-dimension is almost equal. Hence, KNN implementation won't be our consideration.

- Support Vector machine (SVM) is initially to find a hyperplane that can separate two classes, more recent implementations can make use of multiple classes as well [7, 19]. What is the hyperplane? Hyperplane is a mathematical terminology, which decides the boundary between two classes. This boundary can be a point, a line, or a plane if the dimension of data is not more than 3-dimension. Determining the best hyperplane is to maximize the margins from two classes, in other words, the distance between elements of both classes and hyperplane is largest. However, the drawback of SVM is

quite obvious: it is hard to apply for large data due to limitations of the algorithm. Another disadvantage is that, when a dataset contains noises, or when features overlap, the performance of SVM is not satisfactory. In our case, image data is absolute high-dimensional data, among these image datasets, e.g. the shape of histogram and bar chart is pretty similar, both of them belong to the category of column charts. If we apply this method for our datasets, it will be difficult to figure out the difference between those two classes. Therefore, SVM is out of consideration.

- Convolutional neural network (CNN) is the most popular method for dealing with image classification, compared to above methods, CNN will be a better choice for working out our problem. We will explain this choice in the next session in detail.

2.2 Neural Network

2.2.1 Perceptron

Before diving into the concept of CNN, we would like to have a look into Perceptron, which was generic name from biological field [23]. Actually, it is a model of single neuron, in Figure 1 we have input data $X = (x_0, x_1, x_2, \ldots, x_n), n \in \mathbb{N}$, and only one output y, in addition, the perceptron has a bias term 1 as input as well, which is presented as x_0, and every input has its corresponding weight in a linear relationship, which is $W = (w_0, w_1, w_2, \ldots, w_n), n \in \mathbb{N}$ respectively.

$$z = \sum_{i=0}^{n} x_i \cdot w_i \tag{1}$$

The next step is to apply the weighted sum z to the activation function $a(z)$, which is used for determining whether the neuron should be activated or not. Normally the output value of activation function will be 0 or 1: where $a(z) = 0$ means that the neuron is turned off for further learning, conversely, $a(z) = 1$ presents that the neuron is triggered. Activation function can have different types, the most simple example being a step function.

$$a(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \tag{2}$$

With the activation function, the predicted value will be compared with the known output, if they do not match, the weights of z will be adjusted by the backwards propagation of the error. Therefore, the perceptron is able to separate two classes due to binary output. A neural

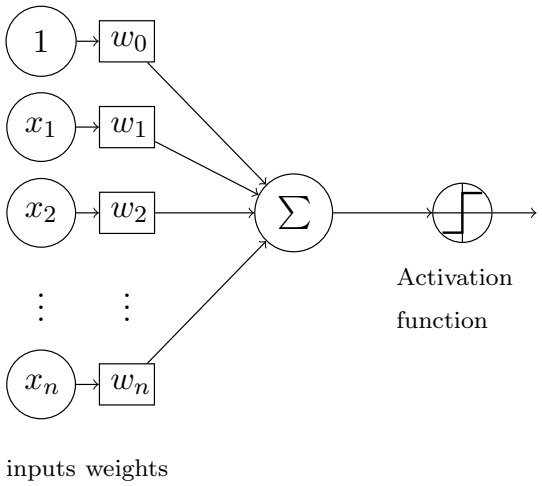


Figure 1: A perceptron with three input and one output

network has multiple computational units just like the perceptron, and will be presented in the next section.

2.2.2 Feedforward Neural Network

Feedforward neural network is a multilayered network of perceptrons, which consists of an input layer, an output layer, as well as multiple hidden layers. An input layer is the initial data for the neural network. An output layer is the final layer of the neural network, which yields the outputs. Hidden layers are the black box in machine learning, which build the bridge to connect the input and output by finding a good implementation. In this neural network, all information goes to one direction, forward, from the input layer through hidden layers to the output layer. Figure 2 is a simple example of feedforward neural network , which has three input variables $X = (x_0, x_1, x_2)$, and two hidden layers with 5 units, of which +1 is the bias unit. We formulate the output as same as above perceptron, the weights for each layer $j-1$ as matrix $W^{(j)}$, where $j = 2, 3, 4$, the activation output $o_i^{(j)}$ is the unit i in layer j, where $i = 1, 2, 3, 4$, we assume that we have M class, here M = 2,

$$x_0 = o_0^{(2)} = o_0^{(3)} = 1 \tag{3}$$

where x_0, $o_0^{(2)}$ and $o_0^{(3)}$ are bias terms,

$$\begin{aligned} z_i^{(2)} &= W_{i0}^{(1)} x_0 + W_{i1}^{(1)} x_1 + W_{i2}^{(1)} x2 \\ &= W_{i0}^{(1)} + W_{i1}^{(1)} x_1 + W_{i2}^{(1)} x_2 \end{aligned} \tag{4}$$

where $z_i^{(2)}$ is the weighted sum in the first hidden layer, i points to the number of units,

$$o_i^{(2)} = a(z_i^{(2)}) \tag{5}$$

where $o_i^{(2)}$ is the output of the first hidden layer,

$$\begin{aligned} z_i^{(j)} &= W_{i0}^{(j-1)} o_0^{(j-1)} + W_{i1}^{(j-1)} o_1^{(j-1)} + W_{i2}^{(j-1)} o_2^{(j-1)} \\ &+ W_{i3}^{(j-1)} o_3^{(j-1)} + W_{i4}^{(j-1)} o_4^{(j-1)}, j > 1 \end{aligned} \tag{6}$$

where $z_i^{(j)}$ is the weighted sum of the i-th unit in the $j-1$ hidden layer, when $j > 1$,

$$o_i^{(j)} = a(z_i^{(j)}) \tag{7}$$

where $o_i^{(j)}$ is the corresponding activated weight sum of the i-th unit in the $j-1$ hidden layer, when $j > 1$, and the vector formalization displays on below:

$$\begin{aligned} X &= O^{(1)} \\ Z^{(j)} &= W^{(j-1)} O^{(j-1)}, j > 1 \\ O^{(j)} &= a(Z^{(j)}), j > 1 \end{aligned} \tag{8}$$

where $O^{(1)}$ is considered as input value, $O^{(j)}$ represents the activated result in each hidden layers, while $j > 1$.

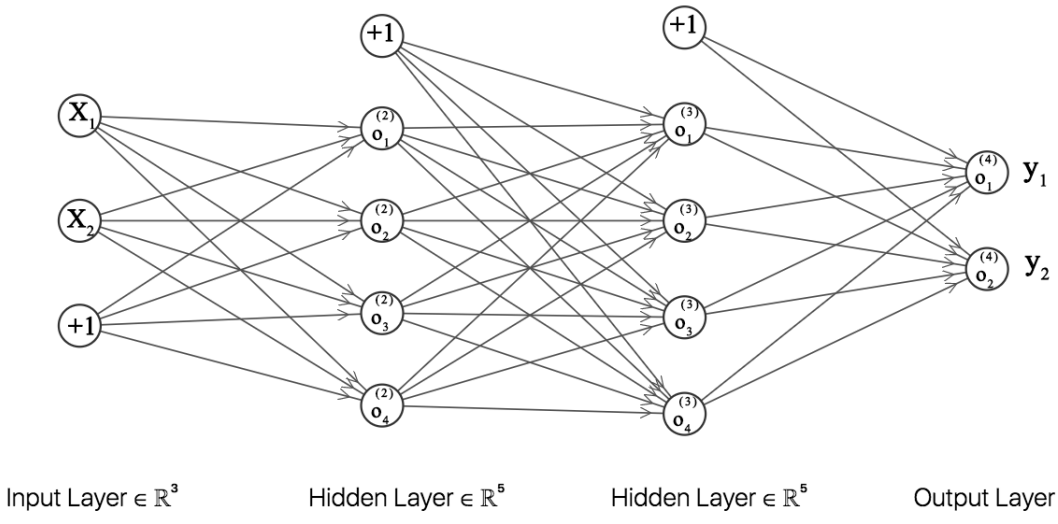


Figure 2: A simple example of Feedforward Neural Network with three input units, two hidden layers of five units and two units in the output layer.

2.2.3 Loss Function

Introducing the loss function is a method of evaluating how good a model does, if the prediction is far away from true value, the loss function attains a very large value. Furthermore, the loss function has an important job in minimizing the error of the model. For example, in linear regression the loss function is the mean square error, i.e. the average on square of distance between the true value and the prediction value. It is intelligible that if the value of the loss function is small, the prediction value will be close to the true value, and the model can be considered as correctly specified, vice versa. The utility of loss function in neural network is the same as in linear regression, there are several types of loss functions in neural network, some of which are the Mean Square Error (MSE), Binary Cross-Entropy, Categorical Cross-Entropy and Sparse Categorical Cross-Entropy respectively.

- MSE: is a typical loss function, which are commonly applies for the linear model. The formula is the average over the square value of the difference between predicted values and true values, we take Fig 2 as the example:

$$\text{MSE} = \frac{(o_1^{(4)} - y_1)^2 + (o_2^{(4)} - y_2)^2}{2} \tag{9}$$

  where $o_1^{(4)}$ and $o_2^{(4)}$ are predictions of y.

- Binary Cross-Entropy: can be also simply considered as logistic loss, which is especially useful in binary classification problems. In binary classification, we have only two values of y, which are either 1 or 0. When $y = 1$ and prediction $= 1$, the loss will be 0. On the contrary, when the true value $y = 1$, while prediction $= 0$, the loss will be a very large number. The formula also takes Fig 2 as the example:

$$\text{Loss} = \begin{cases} -\log(o_i^{(4)}) & \text{if } y_i = 1 \\ -\log(1 - o_i^{(4)}) & \text{if } y_i = 0 \end{cases} \tag{10}$$

  where $o_i^{(4)}$ is the predicted value of y_i.

- Categorical Cross-Entropy: also called softmax loss, widely practices in multi-class classification task. It compares the probability distribution of the true value and the prediction. The formula is similar with binary cross-entropy, if the loss is very small, then the value of corresponding $o_i^{(4)}$ would be very large.

$$\text{Loss} = -\sum_{i}^{M} y_i \log\left(o_i^{(4)}\right) \tag{11}$$

- Sparse Categorical Cross-Entropy: is another kind of Cross-Entropy, the only difference between those two Cross-Entropy formulations is the format of the true value. When the classes are mutually exclusive, using sparse categorical cross-entropy is the better way.

2.2.4 Backpropagation

Backpropagation is an effective method to train a neural network, and the aim of backpropagation is to minimize the loss function by modifying weights and biases. The normal way to minimize a function is to compute the first derivative and set it to zero. But in neural network, the first derivative of the loss function is hard to compute, therefore backpropagation plays a very important role in calculating the minimum of the loss function. Here we take the example from the last section to formularize the backpropagation process.

$$\frac{\partial \text{Loss}}{\partial w_i^j} = \frac{\partial \text{Loss}}{\partial z_i^j} \cdot \frac{\partial z_i^j}{\partial w_i^j} \tag{12}$$

according to the chain rule of the derivatives, where the first derivative of the loss function on weights can be made up of the product of the first derivative of the loss function on the weights sum z_i^j and the first derivative of the weights sum on weights w_i^j,

$$\begin{aligned} z_i^j &= \textstyle\sum_{i=1}^{m} w_i^j o_i^{j-1} \\ \frac{\partial z_i^j}{\partial w_i^j} &= o_i^{j-1} \end{aligned} \tag{13}$$

as we all know, the weighted sum z_i^j is linearly correlated to weights w_i^j, therefore, the first derivative of the weighted sum on weights will be the input value of the i-th unit in the $j-1$ layer.

$$\frac{\partial \text{Loss}}{\partial w_i^j} = \frac{\partial \text{Loss}}{\partial z_i^j} \cdot o_i^{j-1} \tag{14}$$

More specifically, if we want to calculate the weight of the second unit in the second layer $w_2^{(2)}$, the calculation is simplified with backpropagation.

$$\begin{aligned}
\frac{\partial \text{Loss}}{\partial w_2^{(2)}} &= \frac{\partial \text{Loss}}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_2^{(2)}} \\
&= \frac{\partial \text{Loss}}{\partial z_2^{(3)}} \cdot o_2^{(2)} \\
&= \frac{\partial \text{Loss}}{\partial o_2^{(3)}} \cdot \frac{\partial o_2^{(3)}}{\partial z_2^{(3)}} \cdot o_2^{(2)} \\
&= \frac{\partial \text{Loss}}{\partial o_2^{(3)}} \cdot a'\left(z_2^{(3)}\right) \cdot o_2^{(2)}
\end{aligned} \tag{15}$$

2.3 CNN

A CNN is a special kind of neural network, which has been successfully applied for image classification, video recognition, recommendation system, etc. The CNN is also considered as an improved version of a multilayered network, whose elements are similar to the feedforward neural network: input layer, output layer and multiple hidden layers. But the core of any CNN is its convolution operation, i,e. the convolutional layer, which will be described in the following session.

2.3.1 Hidden Layers

Before tackling the convolutional layer, we would like to explain hidden layers, of course, any convolutional layer is a part of hidden layers. Hidden layers in the CNN have different types, which are convolutional layer, pooling layer, fully connected layer, dropout layer and batch normalization respectively.

The convolutional layer is the key element of the CNN and plays a crucial role in extracting the meaningful features. Typically as the first hidden layer in the CNN, it refers to manipulate the input data by a filter kernel, which is a small matrix, called weight matrix. It is used for extraction in a low or high level, and map the appearance to the feature map. Compared to feedforward neural network, main differences are shown below:

- Local Connectivity: an image as a high-dimensional input data, which is unfeasible to fully connect all units of neuron, i.e. only a small local area of images as input. It will help to lessen the burden of memory, and fasten the learning process at the same time.

- Parameter sharing: also called weight sharing, means that same parameters can be applied for multiple units. In the traditional neural network, each component of the weight matrix is only applied once. However, each element of the kernel is sharing for each unit in the layer, futhermore, the sharing parameter highly reduces the amount of parameters. Figure 3 presents how a 3×3 filter kernel applied for a 7×7 input data and obtain a 5×5 feature map in the end, it shows the way of convolution operation in the layer, more precisely, the processing of getting one element of feature map is to element-wise product of input and weight then compute the sum of all elements in this product. Figure 3 also illustrates parameter sharing, as the kernel shifts over the input image with stride = 1, which means that the kernel skips one pixel at a time. On the other hand, the kernel will skip two pixels, if stride is equal to 2.

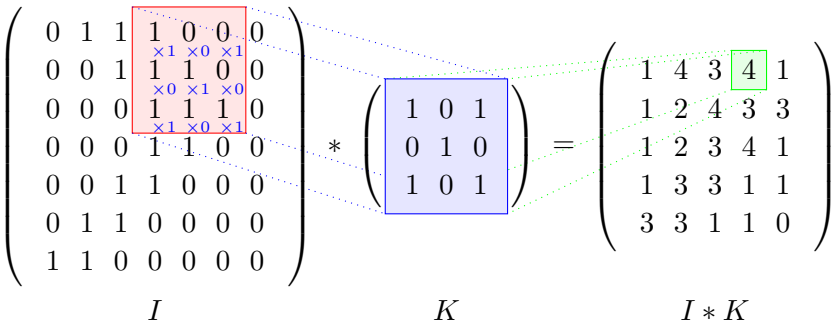


Figure 3: A example of convolutional layer [25]

The pooling layer is normally connected after the convolutional layer, the utility of the pooling layer is to decrease the computational complexity by reducing dimensionality size, which can also avoid overfitting in a way [27]. There are two ways for operating pooling: max pooling and average pooling. Max pooling returns the maximum value of every regime, whose size is identical to the size of filter kernel, which normally is 2×2, while average pooling calculates the average value. Figure 4 illustrates the pooling operation in the layer, it is simple to see that its operation alleviates the spatial size of the input, furthermore, it also reduces the amount of parameters and thus computational overhead. Another useful property of the pooling layer is local translation invariance [10], i.e. the location of a feature will not be changed after applying the pooling layer. The intensity of input matrix from Figure 4 is from 0 to 184, the intensity of the upper part is on average larger than the lower part, i.e. the upper part is brighter than the lower part. After the pooling computation, the location of the bright part is still on the top, and vice versa.

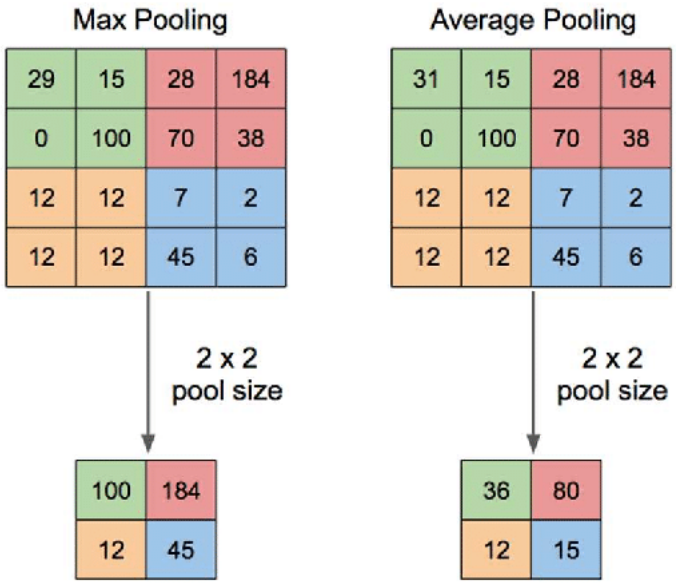
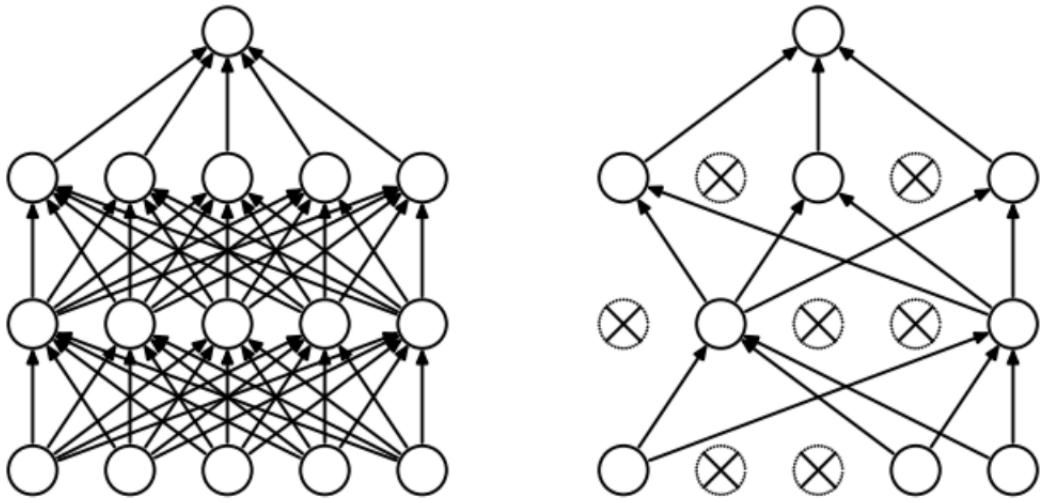


Figure 4: Example of max pooling and average pooling with a 2×2 filter and stride of 2×2[31]

The dropout layer is a special layer compared to above-described layers, and it is a regularization technique for preventing overfitting in the neural network. When training a large CNN without a sufficiently large number of images, the problem of overfitting can not be ignored. Dropout layers pass randomly over some units from layers during the training process and deactivate them. Figure 5 gives a brief description of dropout operation, the dropped units can be any units from the input layer or hidden layers. The hyperparameter of dropout layer is the dropout rate, which is the proportion of training sets that ranges from 1 to 0. There is no dropout, when the value of dropout rate is equal to 1. While if the rate = 0, it means no further connection to next layer or unit. Choosing an optimal dropout rate is also essential [6]: the common setting is 0.5. How to choose the rate for applying a dropout layer will be discussed in the section 4.

Batch normalization is another technique for speeding up the training process and improving performance and stability of the neural network [11]. In general, in order to avoid saturating neurons in the deep neural network, applying with a lower learning rate is a common strategy for training such a model. However, a lower learning rate means longer training time. Hence, a CNN model with batch normalization will improve this phenomenon, even this model with a high learning rate. Another benefit of applying batch normalization in the neural network is its utility of regularization [11]. Batch normalization applies standard normalization for each training batch, more precisely, the output of the previous activation



(a) Standard Neural Net (b) After applying dropout.

Figure 5: A neural model with a dropout layer, here only presents a pattern of dropout. The left picture shows a standard neural network with 2 hidden layers, the right one presents the learning process when applying dropout technique, some units in layers are skipped. [29]

layer is firstly subtracted by its mean value, then it is divided by its standard deviation. This process helps shortening the length of features, therefore, it highly improves the speed of the model's training time.

2.3.2 Activation Function

Choosing the right type of activation function is not only a critical part for the CNN model, but also for all artificial neural networks. It is biologically inspired by the electro-chemical activity in the brain, where whether different neurons are activated or not, depending on the different stimuli. The intuition is to simulate the neuron by a mathematical function for transforming the input, which determines whether trigger next neuron or not. Another property of activation function is a non-linear function and continuously differentiable. If those conditions are not satisfied, the backpropagation would not work. The common activation functions for CNN in multiple classification task are Rectifiere Linear Unit(ReLu) function and Softmax function

- ReLu function becomes commonly used activation function in the CNN after the appearance of AlexNet. It is a simple maximum function, the formula is:

$$a(z) = \max(0, z) = \begin{cases} 0, & \text{if } z < 0 \\ z, & \text{if } z \geq 0 \end{cases} \tag{16}$$

This formula means: if z is a negative value, the function is 0, otherwise, the value of $a(z)$ is z. Due to this simple formula, i.e. the derivative is either 0 or 1, ReLu is less computationally expensive compared to Sigmoid or Tanh function. Glorot et al. [9] showed that ReLu had a better performance for neuronal network, therefore, ReLu with Batch normalization is widely applied for CNN architecture[30, 1, 25, 15], which helps to fasten the convergence of the neural network. However, the disadvantage of ReLu is also explicit: it is not zero centering and has the possibility of killing the unit, which means that the gradient is always 0 and the neuron cannot be activated any more.

- Softmax function, also called normalized exponential function, which presents the probability of multiple class K, will be only applied for the last, i.e. the output layer. The formula of the Softmax function is:

$$a(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}} \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K, \tag{17}$$

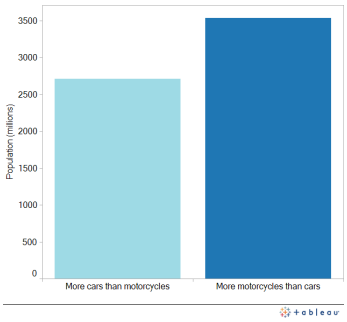
where z_j is the linear combination of weights and input, K is the number of classes, the range of z_j is always from 0 to 1, and the sum of $\sum_{j=1}^{K} a(\mathbf{z})_j = 1$.

3 Data

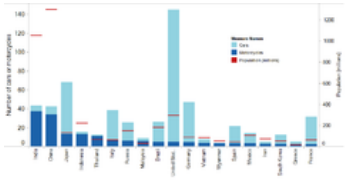
3.1 Data Preparation

As mentioned in section 1, this thesis is an initial attempt to investigate different types of chart images and annotate unlabeled images. All images originate from the Wiki-family, the main part of images is captured directly from Wikimedia Commons and the rest images are downloaded from Wikis of the Humboldt University of Berlin. In Wikimedia Commons, we choose the category called Statistical Charts, in this category there are initially 27 subcategories, even though all subcategories are classified by USPTO, there are still several images in some subcategories that cannot be identified typical chart images. Thus they are not correctly classified by their chart types, instead they have been classified according to their content. For example, there exists a category called 'motorcycle statistical' charts, and subsequently any chart images related to motorcycle can be put into this class. Therefore, bar chart, line chart and pie chart are placed in this class, it would be thus very confusing for the machine to comprehend features from this category. Besides that, the amount of images from some categories are quite small, like the example of motorcycle statistical charts, there are only 4 images, which is not enough information to learn the pattern or structure from those images if we select these images as a training category. Hence, images from those categories with small sample size will not be used as classes during training.

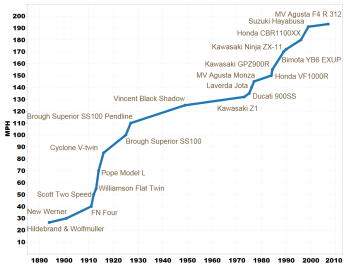
Here we only filter 16 category as our first observations, which are bar chart, box plot, cartogram , correlogram, cumulative frequency distribution (CFD), frequency polygon (FP), heat map, histogram, line chart, pie chart, radar chart, scatterplot, sparkline, stemplot, and violin plot respectively. From Fig 7 it is straightforward to see that some of the different classes are strikingly similar. For example, understood simply as pictures, bar charts are akin to histograms. Likewise, the pictorial representation of line charts, FPs, CFDs and sparklines are visually congruent, even if their respective statistical interpretation could not be farther apart. All images from category Statistical Chart are extracted by Imker, a java-based tool, which was developed by Wikimedia Commons. A total of 3987 images from those 16 classes and 298 unclassified images are scraped, but 3987 images for training a CNN are still a small pool, therefore, we take 262 images from Wikis of the Humboldt University of Berlin, which are used for explaining statistical problems. They are typical statistical chart images, which means that they can be directly classified by categories of Wikimedia Commons. But most images from Wikis of the Humboldt University of Berlin are unlabeled, so we need to manually classify them to their corresponding categories.



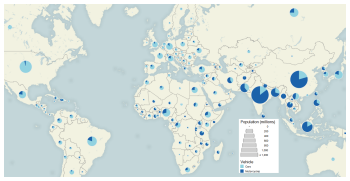
(a) Comparison motorcycles - cars



(b) Comparison motorcycles - cars



(c) Fastest production motorcycle



(d) Fastest production motorcycle

Figure 6: Motorcycle statistical charts

Due to multiple labels in images, we split images with overlapping features into a new category, for example, bar chart with a trending line. In the training process, we observe that plenty of bar charts with a tendency line will be classified into the class of line chart, therefore we create a new category called bar line chart, which contains both, multiple bars and a line. In a sense, it will shrink the potential error and eliminate the influence on overlapping features. Furthermore, due to the small amount of sparkline, and in particular, sparklines are akin to line charts, we merge sparkline images into the class of line chart.

3.2 Image Preprocessing

After the collection of all images from the Wiki family, we find out that the formats of images are quite distinct, so the first step of image preprocessing is to unify the images' formats. In our case we have images in 11 formats, which are tiff, JPG, PNG, xcf, jpeg, WebM, gif, tif, svg and PDF respectively. Many different approaches have been used to deal with this problem. In the beginning, we attempt to apply a python package, called cairosvg, to all samples, but functions of this package only work for converting svg files to PDF or PNG. Therefore, we use a more powerful image processing tool called ImageMagick, which is an open resource software for modifying, editing, converting images. But in those images we

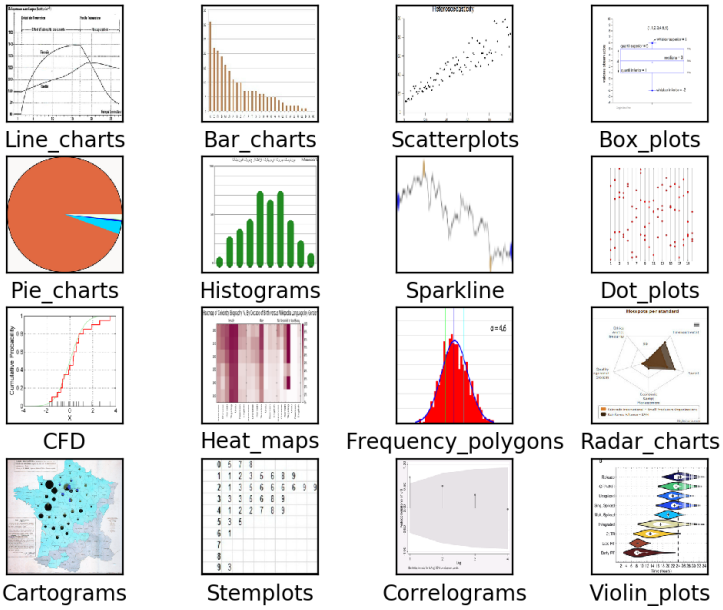


Figure 7: The illustration of chart images from each class

even have images in video format WebM, which cannot directly convert to png or jpeg, and especially, not every frame contains a chart image. Also, extracting practical frames is also an involved process in itself, but luckily we have only 2 WebM files. The rest of images are converted to PNG or JPG. Further attention should be paid when converting GIF files to PNG: it enriches the amount of samples in a way, because all frames in GIF files are relevant to chart images, but it is also crucial to remove duplicated frames to avoid manipulating the error in the further training.

3.2.1 Data Normalization

The second step of image processing is grayscale image and Min-Max normalization, certainly, color is effective information in computer vision, Krizhevsky [14] presented how to extract effective features from the dataset: CIFAR-10, among which color was a fruitful point. However, in our case, color is not mandatory, because it is not the nature of property but rather an artificial feature. Grayscale images for our observations would certainly avoid complexities in the training process, even decrease false classifications. Therefore, we apply the function from python package cv2 to obtain black-white pictures.

In machine learning, normalization is a common strategy for getting a handle on data

in different scales, it is also called data scaling. Before the data analysis, it is vital to verify that all variables in dataset are compatible with each other. On the other hand, data scaling can accelerate the training process due to decreasing the range of data. There are two common strategies for data scaling, which are Min-Max normalization and Z-score normalization respectively.

Min-Max normalization is a normalization strategy to linearly transform data x to range $(0, 1)$, and the idea is to subtract its minimum value of x and the subtraction is divided by the difference of the minimum value of x and the maximum value of x. With this approach, the data can be bounded between 0 and 1. The scale of data will be sharply shrunk, and also will fasten the convergence of the gradient.

$$z = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{18}$$

Z-score normalization is another normalization strategy, in statistics, it is also called standard normalization. The normalized score z is produced by subtracting its mean value and divided by its standard deviation. The scale of x after applying for Z-score normalization will be shrunk in some level, but the resulting range is not always between 0 and 1. However, when the mean value and standard deviation are derived from different algorithms, the score may not be robust [26].

$$z = \frac{x - \mu}{\sigma} \tag{19}$$

where μ is the average value of x, σ is the standard deviation of x.

Another attempt is to remove images' background such as the grid. As we all know, a bar chart is made up of multiple columns, and thus components of a grid can be also seen as different rectangles. Therefore, the background of images may confuse the machine to detect the main body of images. If we are able to remove the grid background, the error rate in the training processing due to eliminating the similarity between column chart and the grid may be lessened. The key point of erasing the background is that the machine is capable of detecting the foreground and background. Generally the color intensity of the background is lower than the intensity of the foreground. Through the image manipulation we can emphasize the foreground and eliminate the background in the end. But in our case, the background of some images even has a distinct advantage of the foreground, then it will be tough to utilize this algorithm to all images. For instance, dash lines in the box plot, which represent the distance between the maximum value and the third quartile, and another distance between the minimum value and the first quartile, are pretty similar to grid-

lines in the background. When we try to erase the background of box plots, such important component of them will be eliminated, that means, removing the background may result in a loss of critical features. Therefore, we cannot simply remove the background for all images. Fig 8 shows comparison between original box plot and processed one.

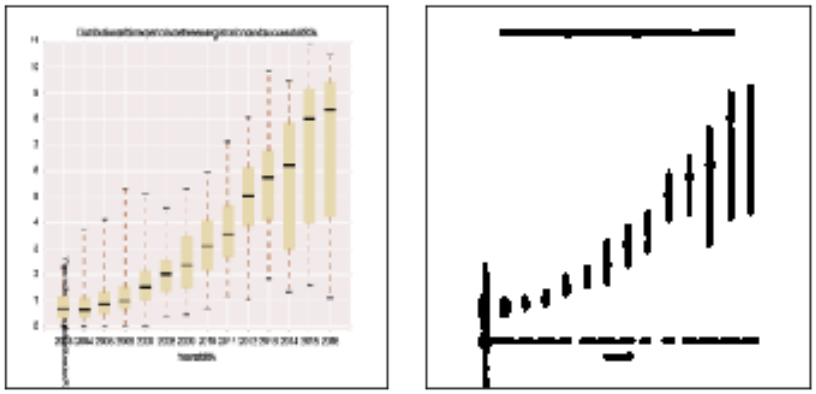


Figure 8: Example of removing background, left picture is the original box plot, right one is removed the background, this process will lose information of data.

Next move is to step deep into the structure of the dataset, and to check if the dataset is balanced or not and also the size of images. It is obvious to detect that classified images are extremely imbalanced, over 1400 images are line chart images, around 1000 are bar chart images, and almost 700 are pie chart images. However, the number of some classes e.g. dot plot, violin plot and sparkline are even less than 20 [1]. Furthermore, the size of images are in a wide range: the minium height is only 20 pixels, on the contrary, the maximum height is up to 15300 pixels; similar in terms of width, the minimum width is 20 pixels as well, but the maximum width is 14000 pixels. The height of most images distributes from 200 to 650 pixels, and width concentrates on the range from 300 to 700, see figure 22 in the appendix for more details.

3.2.2 Oversampling

Handling the imbalanced dataset will be crucial for the further image processing, because an imbalanced dataset can simply get high accuracy, even though the model may be already misspecified. Common approaches to create a balanced dataset are oversampling and undersampling: oversampling is to create new samples that could fulfill the length of majority class

[1] See Table 5 in the appendix for more details

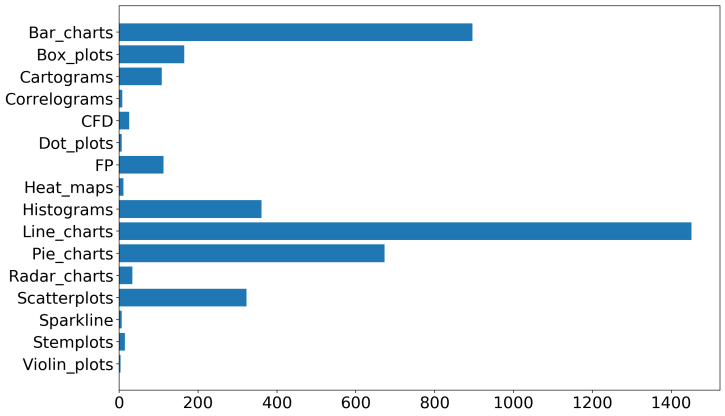


Figure 9: The number of images for each category

samples, in contrast, undersampling is to select samples that could match up to the length of minority classes' samples. In our field the length of minority class such as violin plot, dot plot is just 5, undersampling will discard many of potential features and samples, and it is also tough to train the data with only fewer samples. In addition, Japkowicz and Stephen [12] showed that random oversampling is much more useful than random undersampling. Hence, the major objective of manipulating imbalanced data is oversampling on the minority class.

Here are two common oversampling methods: random oversampling and Synthetic Minority Oversampling Technique (SMOTE). Random oversampling is to randomly select the samples from minority class as supplement, i.e. duplicate images in the minority class. Chawla et al. [5] proposed that SMOTE is another oversampling approach in which the minority class is oversampled by creating "synthetic" examples rather than by oversampling with replacement. This approach would get a better performance on the accuracy of classifier on a minority class.

3.2.3 Data Augmentation

Another usual strategy for handling imbalanced data is data augmentation, which is to increase the variety of images without taking more images. In other word, data augmentation is to modify the original image sightly and thus various of choices of practicing data augmentation can be obtained. With the help of the python package Keras, it is simple to apply data augmentation to the dataset, all parameters are listed on below:

- Rotation: rotate a image at any angel, but rotation at a small angel is more practical

in the most datasets.

- Flip: flip the image horizontally or vertically, vertical flip is same as rotation at 180 degree, however, flip also depends on the type of picture, for example, horizontal flipping language images may not be a good idea.
- Random Noise: randomly add or subtract the pixels with a small number.
- Blur: blur the image by a Gaussian or other function.
- Crop: randomly extract an area from the original image and resize it to original size.
- Contrast change: alter the contrast of the original image with high or low value.

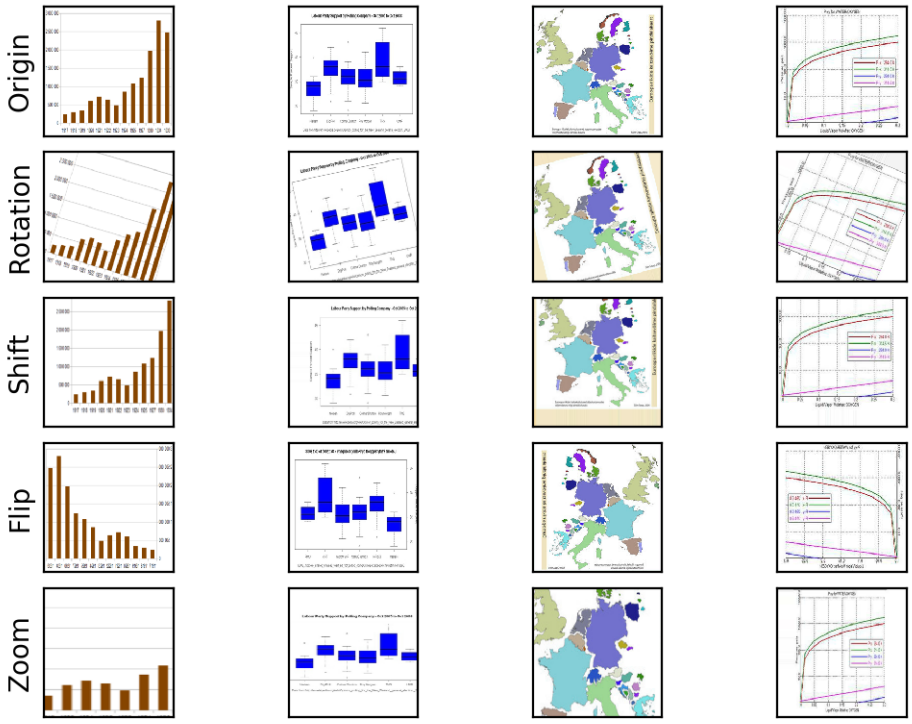


Figure 10: Data augmentation on a sample from bar chart, box plot, cartogram and line chart

For human's eyes, even we flip a little bit on a chart image, we can still recognize it as a chart image, but for the machine, flipped image is a completely new image, since the pixel

information of that image is changed, the location of intensity of pixels are altered as well. Hence, data augmentation is widely applied for deep learning on images. However, how to correctly apply this approach deserve a discussion, a general rule of thumb is to use more choices to change the image and a better performance will be obtained. As for our dataset, all training images are rotated by 30 degree, horizontally flipped and zoomed in, see the details in the Figure 10.

4 Method

4.1 Transfer learning

In this section we will present transfer learning, model architectures and how to tune hyperparameters for enhancing a model performance. As we mentioned before, our dataset is relatively small for training a deep neural network, therefore, there are two methods for training such small database with the CNN model. The first one is to choose a simple CNN architecture, which can be LeNet-5. LeNet-5 as an elementary template of CNN also achieves good results on chart image labeling [2]. The second strategy is to apply transfer learning on the small dataset.

Transfer learning is a machine learning technique that pays attention to store the acquired knowledge from the solution of a problem, and then apply the common part of this knowledge to solve another problem. Humans are good at transferring the knowledge that they have already acquired, to a new field. For example, a badminton player has an advantage of learning how to play tennis: even though badminton rules are different from tennis rules, they are similar to each other. With the help of "transfer learning", this player is able to apply the common sense of the knowledge in playing badminton to play tennis.

Transfer learning is not a foreign approach in latest years, the prototype of transfer learning is the multi-task learning, which attempted to train multiple different tasks at the same time [3]. The core of transfer learning is to extract the knowledge from one or multiple source tasks, and apply it to a target task [21]. The source task for instance can be the knowledge of playing badminton or other racquet sport, the target task is how to play tennis. At present, transfer learning is getting applied in deep learning for text data, computer vision, and audio/ speech recognition, and it is also successfully implemented in deep neural networks such as VGG, which will be introduced in the next session. Here we are only interested in the implementation of transfer learning for deep neural networks, i.e. how to extract common features from the source task, and graft it to the new target task. There are two common transfer learning approaches:

- Fixed extractors of CNN: the last fully connected layer from pre-trained model is replaced for a new dataset, which means, rest layers of pre-trained model are remained, only the last layer will be changed. Due to training less parameters instead of a complete model, the computation will be apparently sped up by training one single layer. It works for small datasets as well, because training a deep neural networks with a small

dataset is quite unfeasible, so training one layer with less parameters may avoid over-fitting. However, the new dataset must somewhat similar to the pre-trained dataset, otherwise, the model performance is not as well as wished.

- Fine-tuning on CNN: it not only replaces the last classifier layer, but also fine-tunes the pre-trained model weights by backpropagation, i.e. block some layers and train the last few layers, then update their model weights by the backpropagation method. Yosinski et al. [33] made an experiment to test the transferability in convolutional neural networks: lower layers can be widely applied for both source data and new dataset, but weights of higher layers only specialize for the source data. If weights of higher layers directly practice in a new target data, the model performance will be discounted by higher layers. But with fine-tuning, model weights of higher layers will be retrained by the target data, thus the model performance will be improved. Hence, the model is more special for the target data rather than the source data.

Here we implement both methods for checking the model's performance. We make use of the pre-trained model VGG-16, which obtained a great achievement on ImageNet challenges [24], it contains more than 14 million images with over 20,000 categories. Even though there is no chart images class in the ImageNet, there are multiple geometric shapes such round, rectangles, which are analogous to our dataset. This is because chart images are made up of some geometric shapes as well. Hence, we can assume that our database is associated with the geometric data from ImageNet.

4.2 CNN Architecture

In the beginning of training task we start to try the simple CNN structure LeNet-5 [16], which consists of 7 hidden layers: 2 convolution layers, 2 max-pooling layers and 3 fully connected layers. The structure is represented in the Figure 11. This architecture is common used as the first initial architecture, which also becomes a standard template. As the example shown in the figure, mass of convolutional layer and pooling layer, connecting one or multiple fully connected layers before the output layer, are successfully applied for recognizing handwritten digits [32]. It has been found that this architecture functions well on small data. The performance of the updated version of LeNet-5 on chart image classification is quite reliable: the accuracy rate is up to 89.5% [2]. The simple structure has a big advantage in time consumption, which is proved that the training time of this architecture is apparently less than other structures. Hence, LeNet-5 is one option for handling our database.

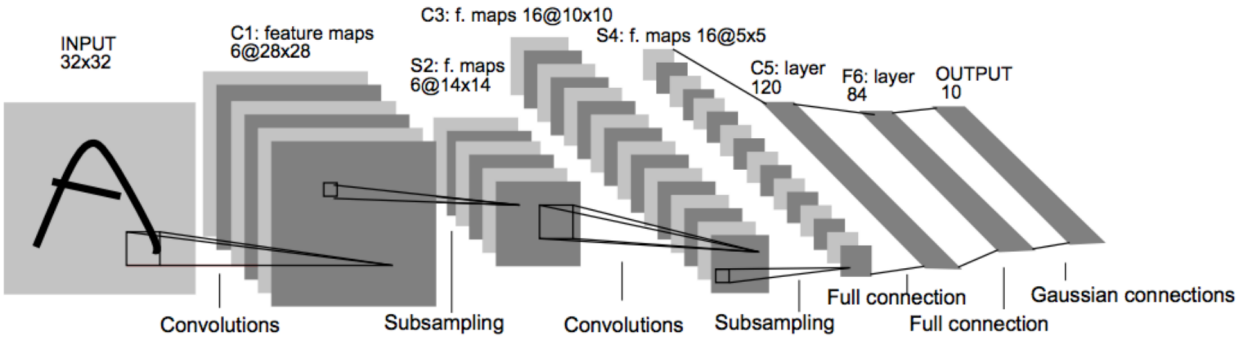


Figure 11: LeNet-5 architecture[16]

The second trail of architecture is AlexNet [15], which can be considered as a deep version of LeNet-5, with the development of computing devices such as graphics processing unit (GPU), which has a great advantage of dealing with image manipulation, due to its parallel structure. Even the complex model can thus be trained with GPUs, of which AlexNet is a good example. The pattern of AlexNet is similar to LeNet: instead of two convolutional layers in LeNet-5, AlexNet stacking 3 groups of convolutional layers and pooling layer. One convolutional layer together with a pooling layer build the first two groups of layers. In the last group of layer, there are three convolutional layers with one pooling layer. Finally the network ends with three fully connected layers. Due to more groups of convolutional layer and pooling layer, the number of parameters in AlexNet is about 61 million, compared to LeNet-5 with around 60,000 parameters. It is not only the revolution of neural networks, but also a great achievement of computing equipment. Furthermore, this architecture is also the first one to use ReLu function as the activation function. This architecture at that time has been considered as one of the largest CNN structures in the ImageNet challenge, which highly improves the performance of CNN on the multiple-class image classification. We will compare the model performance between LeNet-5 and AlexNet in order to select the better model for our dataset.

The last architecture that we will explain is VGG-16, with growing demand of deep learning, the neural network is also becoming deeper and bigger. This means, the amount of hidden layers is expanding: with up to 138 million parameters, it is twice larger than parameters in AlexNet. This model was firstly proposed by Visual Geometry Group from Oxford university [28]. VGG comes from the abbreviation of the group name. Actually, VGG is not a single model architecture, which presents a bunch of architectures designed

by this research group. VGG-16 is widespread in practice, with 21 hidden layers, wined the ImageNet challenge in 2014. The siblings of VGG-16 are VGG-11 and VGG-19 respectively, their differences determines the number of hidden layers. The pattern of VGG-16 presents in the Figure 12, which is distinguished from LeNet-5 and AlexNet: three convolutional layers with kernel size 3×3 connecting with one pooling layer, this set of layers has the same receptive field compared to one convolutional layer with kernel size 7×7 [28]. Theoretically, the larger kernel size is applied, the better model performance will be obtained. However, a large kernel size means more computation in convolution operation. The receptive field is defined as the region of input which is observed for extracting features during convolution operation. The size of respective field relies on parameters of convolutional layer: the kernel size K, the stride size S, the size of the respective field RF_i, the size of the last respective field RF_{i+1}, in which the stride size is equal to 1 [28].

$$RF_i = (RF_{i+1} - 1) \times S + K \tag{20}$$

We assume that the size of final feature map, which operated after three 3×3 convolutional layers, is equal to 1×1, and its respective field is 1 as well. Therefore, followed the equation on below, we can conclude that the respective field after three 3×3 convolutional layers is same as after one 7×7 convolutional layer.

$$\begin{aligned} RF_4 &= 1 \\ RF_3 &= (1 - 1) \times 1 + 3 = 3 \\ RF_2 &= (3 - 1) \times 1 + 3 = 5 \\ RF_1 &= (5 - 1) \times 1 + 3 = 7 \end{aligned}$$

where the kernel size is equal to 3, and corresponding stride is equal to 1.

Therefore, instead of setting a layer with a large kernel size, using this framework with multiple small kernel size layers will speed up the computation, especially for deep neural networks. However, VGG is still a heavy model compared to LeNet-5 and AlexNet, and it is thus considered as a very deep neural network simultaneously.

After the description of VGG architectures, we would like to show the functionality of VGG in transfer learning, as it mentioned in the last session, there are two approaches for transfer learning.

- Fixed extractors of VGG-16: in Python package Keras, users are able to load all weights of VGG-16, and the architecture of VGG-16 in Keras is divided to 5 blocks, 1 flatten

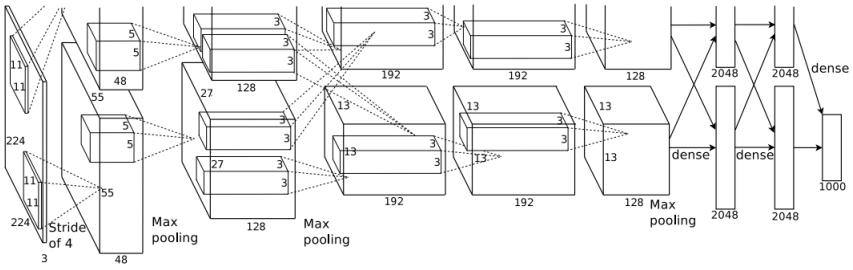


Figure 12: VGG-16 architecture[15]

layer, and 3 fully connected layers. Especially, the last fully connected layer is the classifier layer, which will produce prediction results. Each block refers to the group of convolutional layers and the pooling layer. The transfer learning method is to remove the last three pre-trained fully connected layers, and update the size of the last prediction layer as 16, which is the number of our classes. In the original VGG-16 model, the size of last classifier layer is 1000. The model's weights in 5 blocks are originally from pre-trained model in Keras, only the weights in the last three fully connected layers will be retrained. Check Table 6 to see more details.

- Transfer learning with fine tunning: compared to the method above, much more retrained layers in transfer learning will be fine tuned. The approach is to freeze the first three blocks, and retrain rest of layers, i.e. weights of block 4, block 5 and last three fully connected layers will be renewed by training our dataset. The size of classifier layer is likewise 16. Of course, it is more time-consuming for fine tunning, and the specialty of layers will be closer to our dataset. Table 7 represents the training process of fine-tunning.

4.3 K-Fold Cross Validation

Cross validation is a re-sampling method which is used to evaluate the performance of model, especially on a small dataset. K is the number of groups which is split on the given dataset. The utility of cross validation is to assess the model performance and to get more accurate estimations of parameters. As we all know, loss and accuracy on the training set play a crucial role in deriving parameters. Loss and accuracy on the test data is the core of evaluating the model, when the loss is small while accuracy is high, we can consider this model's performance as a good one, but we are not sure whether parameters by training set are the best or not due

to randomly selection of the training set and validation set. Hence, K-fold cross validation is a sophisticated method for parameter tunning [4], which can help us to find the best parameters by training different partitions of dataset. The core of K-fold cross validation is to split the data into train set and test set, in general, 80% data as training set, 20% data as test set, after that, splitting the training data into the K groups is the next step, further, choosing one of K groups as validation set, rest $K-1$ groups as training set is for training the model, and the best performance on validation set will determine the best parameters. For instance showing in Fig 13, the blue cell stands for validation set, white cells are training sets. Here we can apply those parameters on the test data, which can be considered as the best parameters of data training.

Applying K-fold cross validation for the imbalanced dataset may cause that, samples in subsets of the training set are totally same, especially the dataset is extremely imbalanced. For example, suppose that we deal with the problem of an E-commercial company, which is concerned about the return rate of their customers, normally most customers will not return the item they ordered, but due to some personal reasons, they may return the product. We assume that we have 100 customers, of which 99 customers did not return their items, while only one of them returned the item. It is simple to have $K-1$ groups of samples are the same, i.e. customers did not return the item. In order to fix the problem of the imbalanced dataset, oversampling and undersampling are necessary strategy to build a balanced dataset for further application.

In our case, we apply K-fold cross validation for selecting better weights in CNN, considering the calculation will be set in a loop, which means, training a CNN model will be repeated for K times when applying K-fold cross validation, thus the computation will be very expensive. According to the training devices we have, 10-cores CPUs cannot afford a complex CNN model with the large scale image dataset. Hence, in this method we will only apply for binary classification with the architecture LeNet-5 to confirm the utility of this method. Binary classification is to train the CNN model separately for each majority class: bar chart, cartogram, histogram, line chart, pie chart and scatter plot respectively. For instance, Line chart is one majority class of our dataset, we randomly choose images which are not in the class of line chart, the amount of those images is same as the amount of line chart images(1451). Then we split those 2902 images into train set and validation set, train set is 80% of new line chart dataset, the rest 20% is the validation data, and thus images from the test data will be relabeled as line chart or other chart. We practiced 10-fold cross

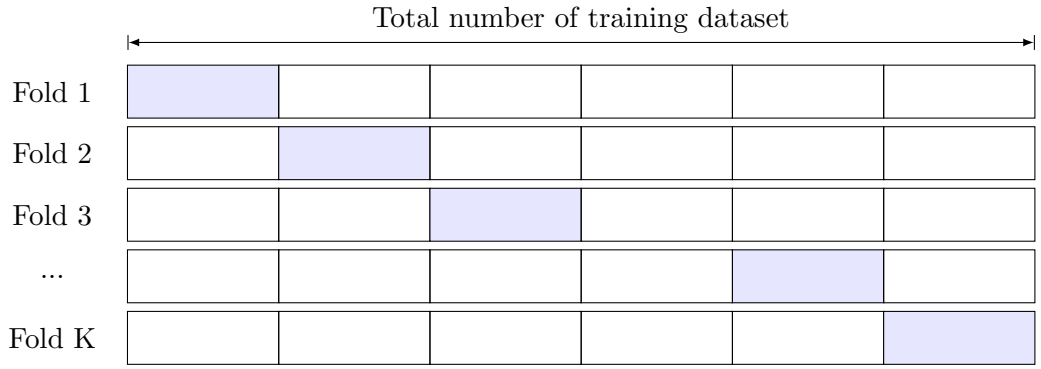


Figure 13: K-Fold Cross Validation

validation on training data set, and it would derive best parameters of LeNet-5 for training binary case of line chart.

4.4 Model Selection

Model selection is the process of choosing the best performed model from all possible models for further prediction, and it also includes the selection of hyperparameters. All candidate models are three CNN architectures, therefore, tuning hyperparameters is the key of CNN model selection as well. Selection of the CNN structure is done by comparing their test accuracies and losses on the test data with different CNN structures. In Python we build a model function to help us for selecting the CNN structure by iterating over all hyperparameters for those three architectures.

Hyperparameters are the key point of model selection, however, their value is set before the model training, therefore, choosing a good interval of hyperparameters is also as important as training weights by reducing the loss. A general method for selecting hyperparameter is grid search, we list all possible values of hyperparameter, iterate over the hyperparameters applied to the model, and select the best values by the test accuracy and loss. The CNN relies on the value of the hyperparameters, therefore, we tune hyperparameters from table 1.

where the learning rate plays a crucial role in optimization of the loss function and controlling the speed of training process. If its value is too small, it will take more time to converge or get stuck on suboptimal point. On the contrary, if its value is too large, it may skip the optimal point, and fail to converge. The input size on the one hand relies on the quality of images, on the other hand, it would be better to choose a large scaled image instead of small one, which helps the machine with learning more details from images. However, larger images require more computation power, even though the size of most of our images is over

	Hyperspace
Learning rate	$[10^{-3}, 10^{-4}, 10^{-5}]$
Input size	[32, 64, 128, 256]
Batch size	[32, 64, 128]
Kernel size	[3, 5, 7]
Dense	[256, 512, 1024, 2048]
L2 rate	$[0.01, 0.001, 0.0001]$
Dropout	uniform(0,1)

Table 1: The list of hyperparameters

1000×1000, considering the learning time, we start to train the model with image size 32×32. The kernel size determines the size of filters in the convolutional layer, the rule of thumb for choosing kernel size starts from 3, but larger kernel size means more expensive computations, therefore, we restrict the kernel size to a moderate value.

The L2 rate refers to weights of L2 regularization, which is a common technique for avoiding model overfitting. The format of L2 is the sum of squared parameters W with scale λ, and $\lambda > 0$. $l(\mathrm{W}, X)$ is the loss of weights W and input variable X.

$$L(\mathrm{W}, X) = l(\mathrm{W}, X) + \frac{1}{2}\lambda \|\mathrm{W}\|^2 \tag{21}$$

In gradient descent, weights W will be updated, α is the learning rate, $\alpha > 0$.

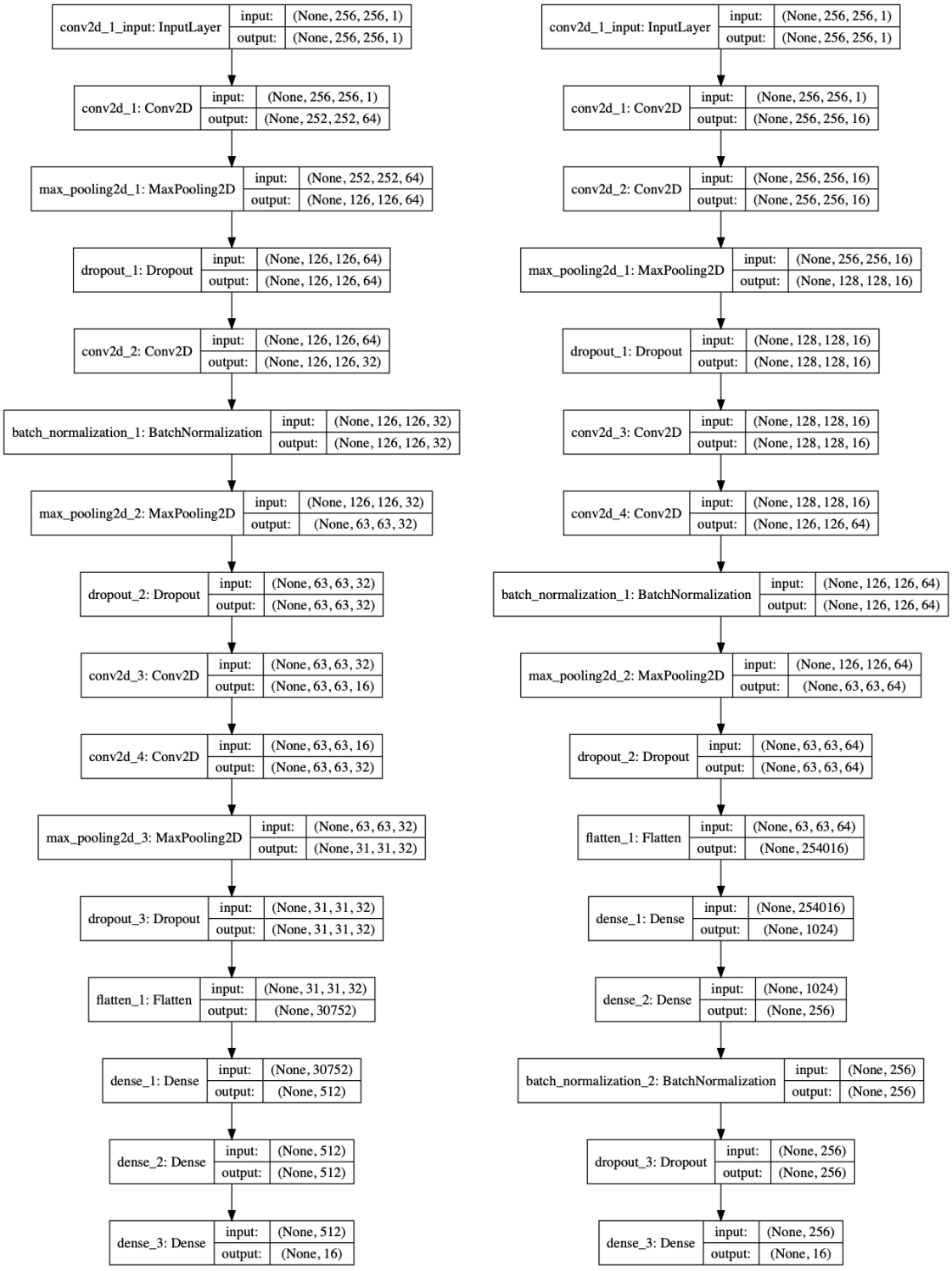
$$\begin{aligned} \mathrm{W} &:= \mathrm{W} - \alpha \frac{\mathrm{d}}{\mathrm{dW}} L(\mathrm{W}, X) \\ &= \mathrm{W} - \alpha \frac{\mathrm{d}}{\mathrm{dW}} \left(l(\mathrm{W}, \mathrm{X}) + \lambda \times \frac{1}{2} \|\mathrm{W}\|^2 \right) \\ &= \mathrm{W} - \alpha \frac{\mathrm{d}l(\mathrm{W}, \mathrm{X})}{\mathrm{dW}} - \alpha\lambda\mathrm{W} \end{aligned} \tag{22}$$

The penalty term $\frac{1}{2}\lambda\|\mathrm{W}\|^2$ will squeeze the value of weights to be relatively small, when the value of parameter is large. Of course, the penalty term also depends on the value of λ, if the value of λ is too large, all weights will be close to 0, the model will be underfitting. On the contrary, when λ is too small, the effect of penalty will be discounted. Therefore, choosing a right λ is also important, thus we apply cross validation for finding an appropriate one.

Considering the size of the dataset, we design two model architectures, which refer to above large CNN architectures: AlexNet and VGG-16. The highlight of AlexNet is the combinations of over one convolutional layer and one pooling layer, which ensures the efficient

extraction of features. This stack of convolutional layer and pooling layer, is also called Alex blocks. The Alex blocks architecture is similar with AlexNet, only the last stack of convolutional layer and pooling layer is different. Therefore, the number of parameters is thus less than AlexNet, and its simple architecture will remedy overfitting. VGG-16 has more combinations of two convolutional layers and one pooling layer, here we applied first two blocks of VGG-16 as our model architecture, which are also called VGG blocks. VGG blocks architecture is similar with VGG structure, which consists of two groups of two convolutional layers, one pooling layer, and three fully connected layers. Due to the small training dataset, we do not design a very complex CNN architecture for training our data. Furthermore, the complex model will easily lead the overfitting problem when dealing with a small dataset. Besides that, adding batch normalization into the model architecture will be very helpful to normalize the feature map from last hidden layer, and speed up the learning process. Hence, our model architecture represents in the Figure 14.

Normally, in Keras it requires four dimensional inputs, which are batch size, height, width and depth respectively, where height, width and depth are the dimension of image, the size of our training images is 256×256, its depth is 1, because images are grayscale. Furthermore, the kernel size of first and third convolutional layer is 3, and the kernel size of the rest convolutional layers is 5. The dropout rate is around 0.5 and 0.7.



(a) Alex blocks

(b) VGG blocks

Figure 14: Our model architectures, left one denotes Alex blocks, right one refers to VGG blocks

5 Results

5.1 Training Phase

This section will evaluate the performance of our models to predict unlabeled chart images. We will compare the LeNet-5 and two transfer learning methods, which were described in the last section. Evaluating the effectiveness of model is according to accuracy, recall, precision and F1 score. Accuracy works well when the dataset is balanced, but if the dataset is imbalanced, F1 score will play a more important role in assessing the model's performance. Accuracy measures the proportion of correct classifications from all of test observations, therefore, the larger the accuracy is, the more predictions are classified correctly. Precision and recall are reflected the accuracy of a model as well. Precision measures the percentage of correct positive classifications from all actual positive samples. Recall denotes the proportion of correct positive classifications from all predicted positive samples. The F1 score is the harmonic mean value of precision and recall, due to the property of harmonic mean value, the smaller value can be emphasized. They are formulated as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{23}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{24}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{25}$$

$$F_1 = 2 \times \frac{\text{precision} \ \times \ \text{recall}}{\text{precision} \ + \ \text{recall}} \tag{26}$$

where TP (True Positive) presents a case of positive sample which is predicted as positive, for example, a line chart is correctly classified into the line chart class. TN (True Negative) denotes a negative sample, which is correctly predicted as negative, e.g. a non-linear chart such as histogram is predicted as a histogram. FP(False Positive) refers to a negative sample, whose prediction is positive, for example, a histogram is classified into a linear chart. FN(False Negative) means a positive sample, whose prediction is negative, e.g. a line chart is sorted into other non-linear chart.

Another tool of model measurement is the confusion matrix, as a specific table, whose rows represent prediction results on each class, while columns display actual results of samples on each class. The diagonal of confusion matrix are the correctly classified samples, which

are TP and TN. On the other hand, the rest of cells refers to FP and FN. For example, a confusion matrix in binary classification is illustrated in Table 2. The confusion matrix summaries the comparison to true labels of images and corresponding predicted labels. It is easy to identify the confusion between classes, which represents not only the amount of misclassified images, but more importantly it also displays where the error comes from and which type of error they are. Its default format works for binary classes, and now it has been applied for multi-class problem. The test result will be shown by confusion matrix in the next section.

	Class A	Class B
Class A	TP	FP
Class B	FN	TN

Table 2: A example of confusion matrix in binary classification case.

In the previous section, as we mentioned, transfer learning is capable of doing a great job of dealing with image classification on small dataset. The training process of the first approach of transfer learning presents in the Figure 15. In this training process we apply random oversampling to each training batch, which makes sure that in every batch the model is trained over the balanced subset. From the figure, it is obvious to see that regardless of the training dataset or on validation dataset, their corresponding accuracies are all up to 0.8, even though the training epoch is not very large. Furthermore, the validation accuracy fluctuates around 0.84 after epoch = 5, and it is apparent to see that there is a gap between validation and training accuracy, which indicates that the model is overfitting. The validation loss varies with epoch, which means, the validation is not converged. Therefore, the final model with only 5 epochs is most suited for overfitting.

Observing on the second method of transfer learning is shown in the Figure 16. The validation accuracy approaches to 0.85 after only one training epoch, and the training accuracy is even up to 0.97. The effectiveness of transfer learning with fine tunning is therefore clear. Furthermore, the tendency of the validation accuracy is similar with the above one, the validation set stops learning since epoch = 2, besides that, from a gap between training accuracy and validation accuracy, it can conclude that the model is overfitting. However, it is reasonable to obtain an overfitting model in the training process, the number of parameters in VGG is almost three times larger than our model, massive parameters are only trained by thousands of images, overfitting in transfer learning with a large epoch cannot be avoided.

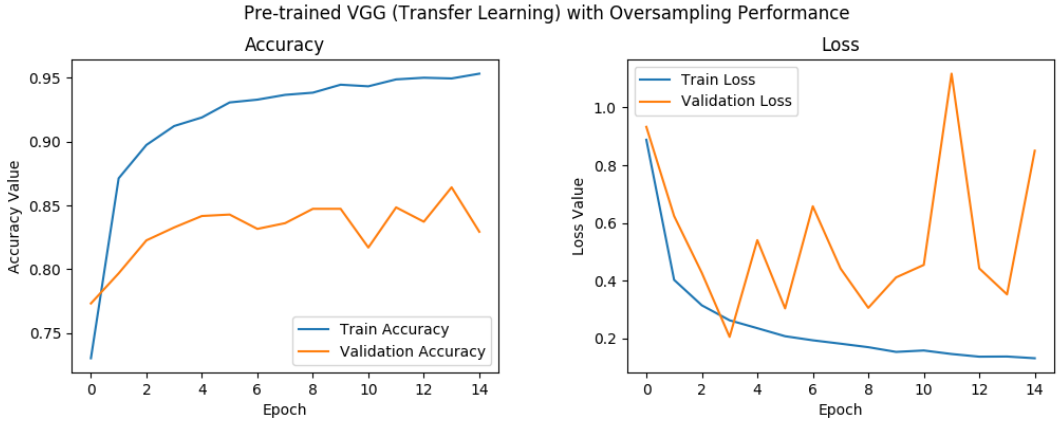


Figure 15: The accuracy/Loss in transfer learning with oversampling.

Hence, training transfer learning algorithm with only 5 epoch in the end, is a considerable solution against overfitting.

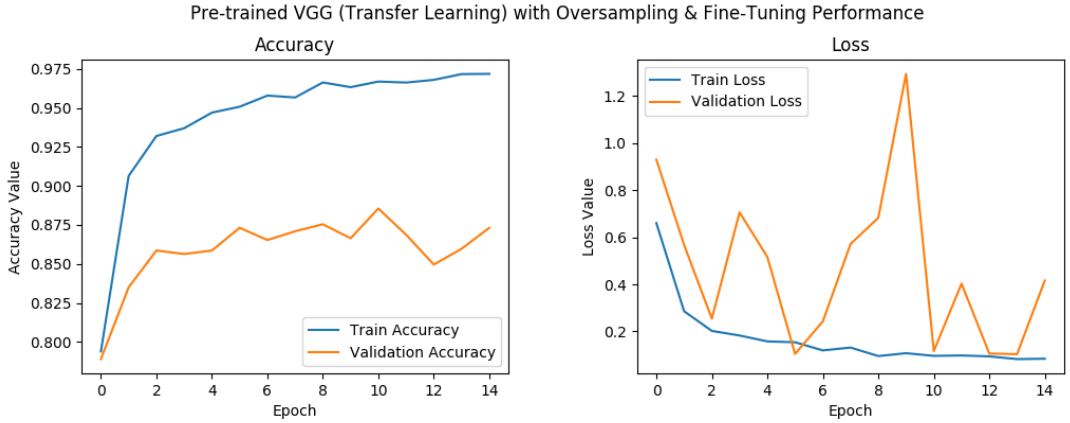


Figure 16: The accuracy/Loss in transfer learning with fine-tuning and oversampling.

The training process of Alex blocks architecture is different with transfer learning. First of all, Alex blocks did not reach such high accuracy in the training process. The largest validation accuracy is around 0.7, and the training accuracy approaches to about 0.9. However, it is clear to notice the gap between the training and validation accuracy, therefore, overfitting is not avoidable, even though Alex blocks architecture is easier than AlexNet. Losses of the training and validation dataset are both decreasing, that means, the loss is converging to the optimal point. Furthermore, the tendency of loss is fluctuating around 1, which implies, the model cannot learn anything new from the validation data.

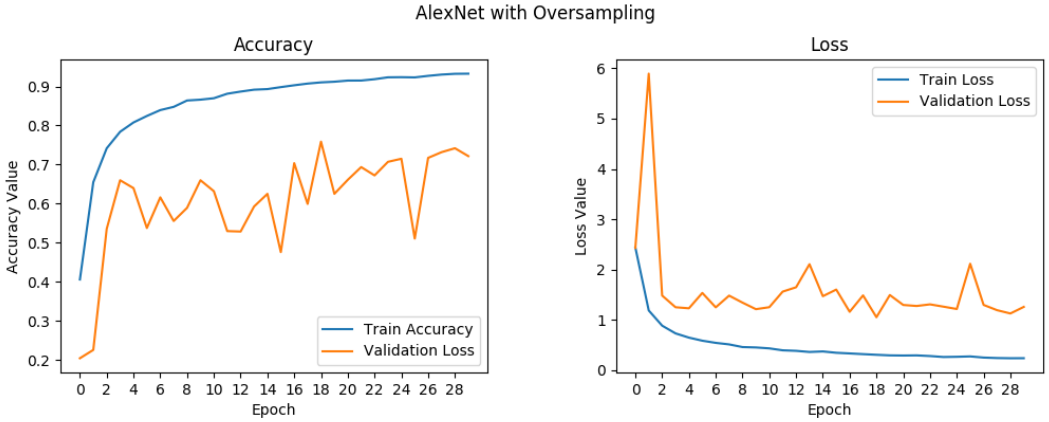


Figure 17: The accuracy/Loss of Alex blocks

5.2 Test Phase

Like the training data, our test data is extremely imbalanced, overall 224 chart images distribute only over 9 classes, which are bar line chart, bar chart, cartogram, CFD, FP, Histogram, line chart, pie chart, and scatter plot respectively. Table 3 represents the frequency on each class of the test data, where the category of line chart is still the majority class in the test dataset, and the class of scatterplot can be also considered as the majority class.

	Nr. of images
Bar line chart	4
Bar chart	17
Cartogram	8
CFD	1
FP	31
Histogram	2
Line chart	88
Pie chart	3
Scatterplot	70

Table 3: The number of images on each category in the test data.

The overall summary of model performance on the test dataset is represented in Table 4. All self-learning models such as LeNet-5, Alex blocks and VGG blocks are applied by the grid search to tune hyperparameters. When practicing transfer learning to our test dataset, grayscale images have to be converted to RGB images, due to the fixed dimension of pre-trained model, whose depth is 3 channel. Hence, adjusting our dataset dimension to match

with pre-trained dataset is necessary. The model performance of transfer learning is clearly better than self-training model, the accuracy on the test dataset with both methods of transfer learning is around 0.6, on the another hand, the accuracy with self-training model is only around 0.4. As it is mentioned before, the criterion of testing on imbalanced dataset cannot be accuracy. However, the precision, recall and F1 score with transfer learning is higher than any one of those self-training models. Even though the accuracy and F1 score of Alex blocks is highest in the self-training model themselves, it is still lower than transfer learning. On the contrary, the precision of LeNet-5 architecture is higher than other self-training model, which means more positive samples compared to other two self-learning models, are correctly classified.

Method	Accuracy	Precision	Recall	F1
Transfer learning on VGG with oversampling	0.63	0.74	0.63	0.64
Transfer learning on VGG with fine-tunning and oversampling	0.65	0.77	0.65	0.63
LeNet-5 with oversampling	0.40	0.73	0.40	0.48
VGG blocks with oversampling	0.42	0.49	0.42	0.34
Alex blocks with oversampling	0.47	0.61	0.47	0.50

Table 4: The summary of model performance on test data

where precision, recall and F1 score are its corresponding weighted average respectively. In the last section, precision as the proportion of correct positive classification from all actual positive observations. In binary classification, once a positive classification is defined by one of those two classes, the other one can be seen as the negative classification. But in multiple classification, if a class is considered as a positive classification, the other rest of classes will be all seen as negative classifications. Therefore, the precision for each individual class can be calculated, and the precision for the entire classification can thus be obtained by introducing a weighted average value, where the weight is the number of images in each classification. Computing overall recall and F1 score is same as the overall precision.

The confusion matrix is a tool to represent details of prediction results, which also plays a great role in identifying the origin of error. The confusion matrix of transfer learning with fine tunning and oversampling represent in the Figure 18. It is obvious to detect at least half of images are correctly classified, expect images from FP and CFD class. The predictions of FP and CFD images are mainly classified as line chart, as our perspective in the beginning, the similarity among FP, CFD and line chart do confuse the machine to

identify their differences, here we can conclude that this approach fails to detect those three chart images. In particular, all bar line images are correctly classified, which means a single trending line is a key information for recognizing the difference between the normal bar chart and bar line chart. It is the reason why we split some bar chart with line into a new class, even though there are 18% of bar charts which are classified to bar line chart. Another point need to be attended: most misclassified images are sorted into line chart class, which provide the evidence of influence of training imbalanced dataset, line chart is the majority class in our dataset, 71% of FP images are considered as line chart images, all predictions of CDF image is line chart, 33% of scatter plot is classified to line chart as well. As it preciously mentioned, some CFD, FP and scatter plot images can be considered as a special line chart, then how to identify differences among those images is our goal.

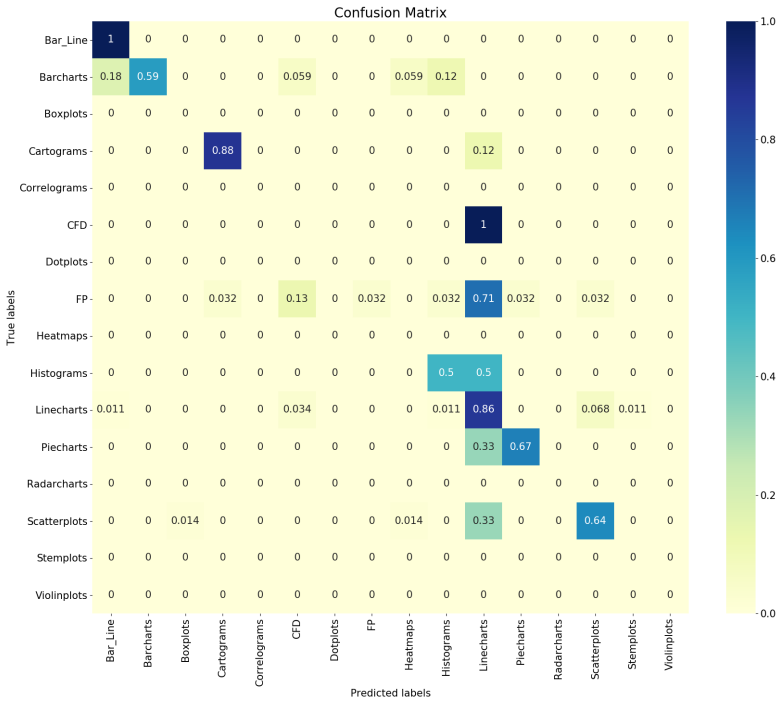


Figure 18: The confusion matrix of transfer learning on VGG with fine tuning and oversampling

Compare to the confusion matrix of transfer learning with oversampling in the Figure 19, nearly half pictures are classified in a correct way, but images from FP, bar line chart are almost misclassified, furthermore, those images are classified to the class of line chart. From this confusion matrix, it is also simple to recognize that for histograms and bar charts,

regardless of which type of bar chart, they are tough to be identified by the machine. 12% of bar chart images are classified to histogram, 25% bar line chart images have same issue as normal bar chart images. However, there is no bar line chart images being predicted as bar chart. For images from Cartogram and CFD are all classified in its true class. However, 23% of FP images are sorted into the class of CDF, but most predictions of FP images are line chart, same issue happens in the normal approach of transfer learning. An interesting point is that 13% of scatter plot are considered as box plot, which is not as we expected.

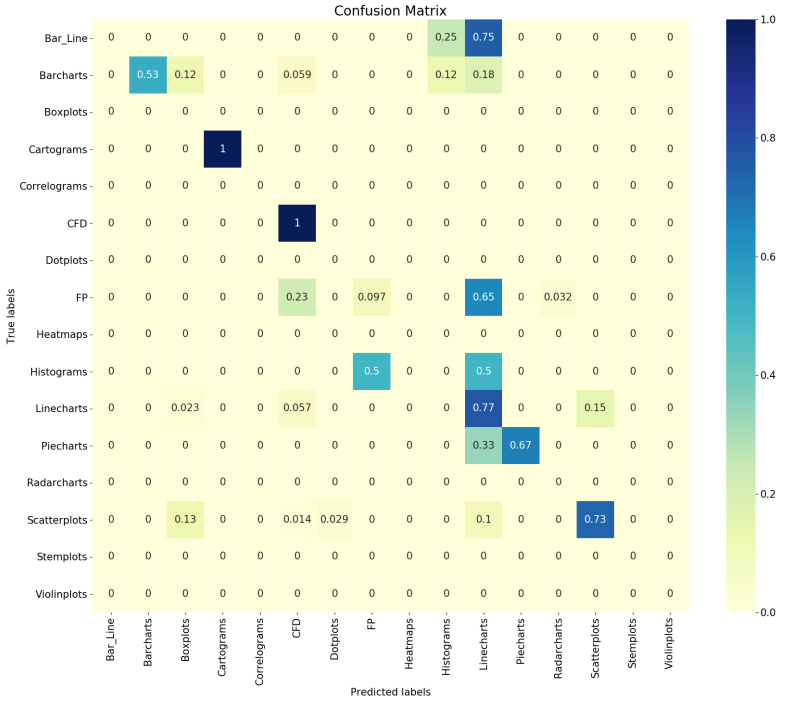


Figure 19: The confusion matrix of transfer learning on VGG with oversampling

The last confusion matrix is the prediction results of Alex blocks, which are displayed in the Figure 20. The prediction accuracy is worse than the prediction with transfer learning, but it is sightly improved compared to other self-training model. This model architecture achieves the goal of identifying differences from bar line chart and bar chart. Furthermore, all histogram and CFD images are classified in correct way. However, this architecture cannot recognize cartograms and pie charts correctly, and FP images look like they are classified randomly to each class, this also happened in labeling line chart images. Alex blocks architecture also has the same issue that most of misclassifications are line chart images. In particular, 50% of scatter plot, 32% of FP and 21% of scatter plot images are

misclassified. Unfortunately, the prediction of pie chart is not as expected, only 1/3 pie chart is annotated correctly, while another 1/3 is classified to histogram images, which is out of the blue, and rest images are labeled to line chart images.

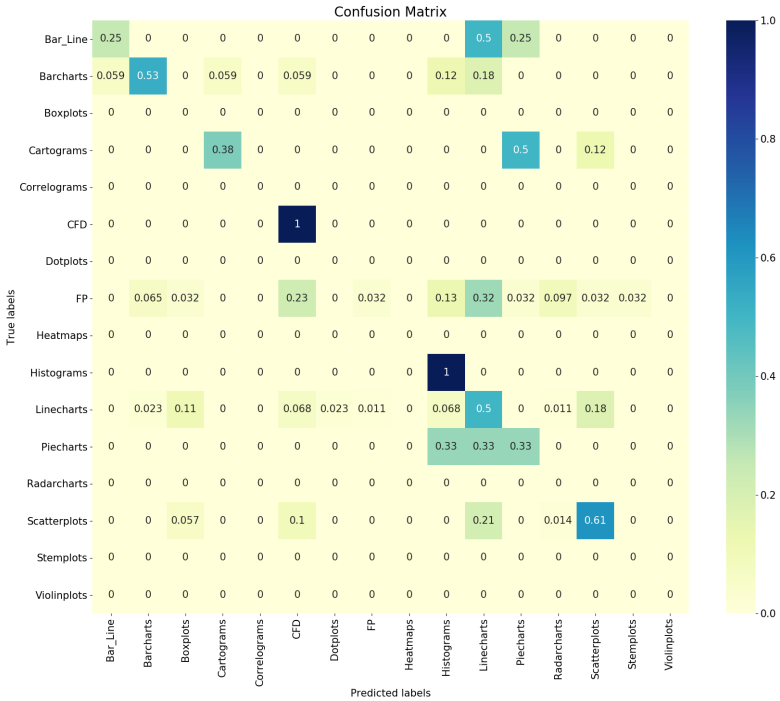


Figure 20: The confusion matrix of Alex blocks

5.3 Binary Classification with K-Fold Cross Validation

As it is described in the last section, we applied the 10-fold cross validation to an artificial balanced binary classification. Those binary classes are based on majority classes in our dataset, such as line chart, bar chart and etc. Test dataset is still the set of unlabeled images from Wikimedia Commons, therefore, test dataset is still imbalanced as shown in the last session, but their labels are edited for binary case already: specific chart and other chart. The test accuracy of each class displays in the Figure 21.

It is simple to grasp that model performance with 10-fold cross validation is better than performance without cross validation. Furthermore, the test accuracy for each class is over than 0.6, especially, the test accuracy in pie chart approaches almost 1, which means almost all pie chart images are correctly classified, and it is also reasonable to obtain this result, the shape and pattern of pie charts are unique compared to other chart images, the round

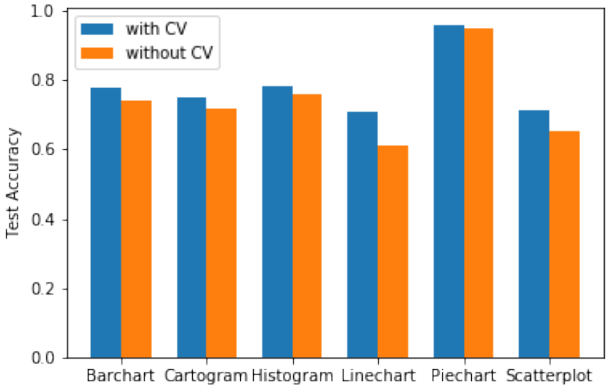


Figure 21: The test accuracy comparison between model with CV and without CV

appearance makes pie chart easily distinguish from other class. The model performance on line chart images is slightly poorer than other classes, which also reflects that labelling line chart is more difficult than other types of images. Line charts are such general charts, that are easily confused with other charts, for instance, scatter plots.

However, the training time of model with cross validation is at least 10 times more than model without cross validation, even though the model performance is more or less improved by 10-fold cross validation, the computation time on binary classification is highly increased. If we apply this method to overall dataset, the computation will more expensive than the binary case. As all we know, k-fold cross validation is well practice in model with small parameters, e.g. linear regression, logistic regression for tunning the best hyperparameters. However, the training time for a model such as convolutional neural network with huge parameters is much more than linear regression. Furthermore, if the training process do not have the support of GPUs, the training session may be killed. In practice, k-fold cross validation is not a optimal choice for training a CNN model [17].

6 Conclusions

In this thesis, our goal is to help Wikimedia Commons to annotate unlabeled chart images by training limited numbers of labeled image dataset. In our analysis, we point out problems with chart images, which are the similarity of chart images, unformatted images, and particularly the issue of this extremely imbalanced dataset. We apply random oversampling on the training set for ensuring balanced training batches, unify the format of images by ImageMagick and grayscale images to simplify the dimensionality of images. Furthermore, in order to obtain better prediction results, we divided bar chart images into bar line chart and normal bar chart images, which guarantees the independence between those two classes, and it can eliminate their relevance to reduce the error rate.

We implement different CNN architectures for our dataset, followed by two directions of CNN methods: transfer learning and self-training, self-training means all weights in hidden layers are trained by architectures themselves, and hyperparameters are tuned by the grid search. Diving into performances of models from the last section, we can conclude that prediction with transfer learning algorithm is better than our self-training models, the test accuracy is improved by around 20%, and values of other measurements such as F1 score are also higher than self-training models. Furthermore, differences between bar charts and histograms are detected by transfer learning with fine tunning, and especially, cartogram images are mainly correct labeled.

Even though self-learning models' performances are a bit worse than transfer learning, the test accuracy can still be around 0.5, which means almost half of test samples are annotated in a correct way. In self-training models, the similarity between chart images and histograms is failed identifying by model. Another common issue in transfer learning come up with self-training model as well: model is not able to detect the differences from PF, CFD and line chart images. The last source of confusion lies between cartograms and pie chart images, half of cartogram images are labeled into pie chart class, which is unintuitive from our perspective.

Therefore, we can conclude that transfer learning with fine tuning is the best implementation of our dataset, which also proves advantages of transfer learning: less time consumption, higher accuracy on small dataset. Transfer learning gives a new perspective of training a complex model with small dataset. Based on pre-trained model, a new target dataset can be easily resolved. Transfer learning offers an opportunity to analyze a new target data without thousands of trails.

In our case, self-training models even take more time than transfer learning, the prediction

results is not such satisfactory, but the self-training model is still the mainstream approach for the general dataset. Here we list some reasons to explain why performance with self-training model is on the average level worse than the one of transfer learning. As we mentioned in previous sections, GPU can highly improve the computation speed of certain algorithms, but due to the lack of GPU devices during our project, most of the models are trained by CPU computation, thus the efficiency of training is highly discounted. When the size of images are over 64×64, the time of training one epoch is over 2000 seconds, i.e. more than 30 mins. Even though the amount of epochs is not very large, but 10-fold cross validation makes this training process more computationally expensive. Secondly, too much noise in our dataset: annotating unlabeled chart images is our goal of this analysis, but if one image contains two types of chart images, it will definitely confuse about extracting significant features from images. Furthermore, the definitions of some classes in Wikimedia Commons determined by the meaning of plot instead of the type of chart. For instance, PF images under a specific scene can be considered as a type of histograms or even line charts, CFD images are able to be identified as special line charts. Most misclassifications origin from the predictions of those three classes. The final one is small dataset which is a common shortcoming of implementation CNN models, As we all know, small dataset is very common in the reality, but training a CNN model with a small dataset is doubtless to encounter overfitting. In our case, violin plot images with only four samples in training dataset will be very difficult to understand its insight.

All in all, regardless of transfer learning or self-training models, dealing with a small imbalanced dataset is the common issue for the classification task. Hence, our research give the perspective of training such kind of dataset, if further improvements can be introduced that overcome drawbacks we mentioned above, then better results can be obtained.

References

[1] Agarap, A. F. (2018): "Deep Learning using Rectified Linear Units (ReLU)," .

[2] Amara, J., P. Kaur, M. Owonibi, and B. Bouaziz (2017): "Convolutional Neural Network Based Chart Image Classification," .

[3] Caruana, R. (1997): "Multitask learning," *Machine learning*, 28, 41–75.

[4] Cawley, G. C. and N. L. C. Talbot (2010): "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation," *Journal of Machine Learning Research*, 11, 2079–2107.

[5] Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer (2011): "SMOTE: Synthetic Minority Over-sampling Technique," *CoRR*, abs/1106.1813.

[6] Dahl, G. E., T. N. Sainath, and G. E. Hinton (2013): "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 8609–8613.

[7] Foody, G. M. and A. Mathur (2004): "A relative evaluation of multiclass image classification by support vector machines," *IEEE Transactions on Geoscience and Remote Sensing*, 42, 1335–1343.

[8] Fukushima, K. (1980): "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics*, 3, 193–202.

[9] Glorot, X., A. Bordes, and Y. Bengio (2011): "Deep Sparse Rectifier Neural Networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ed. by G. Gordon, D. Dunson, and M. Dudík, Fort Lauderdale, FL, USA: PMLR, vol. 15 of *Proceedings of Machine Learning Research*, 315–323.

[10] Ian Goodfellow, Yoshua Bengio, A. C. (2016): *Deep Learning*, THE MIT PRESS.

[11] Ioffe, S. and C. Szegedy (2015): "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *CoRR*, abs/1502.03167.

[12] Japkowicz, N. and S. Stephen (2002): "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, 429–449.

[13] Jung, D., W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo (2017): "ChartSense: Interactive Data Extraction from Chart Images," 6706–6717.

[14] Krizhevsky, A. (2009): "Learning multiple layers of features from tiny images," Tech. rep.

[15] Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012): "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Curran Associates, Inc., 1097–1105.

[16] Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998): "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 86, 2278 – 2324.

[17] Li, F.-F. (2017): "Lecture notes in Convolutional Neural Networks for Visual Recognition," Available at `https://cs231n.github.io/classification/#val`.

[18] Marr, D. (1982): *Vision – A Computational Investigation into the Human Representation and Processing of Visual Information*, THE MIT PRESS.

[19] Mayoraz, E. and E. Alpaydin (1999): "Support Vector Machines for Multi-class Classification." vol. 2, 833–842.

[20] N. Kouiroukidis, G. E. (2011): "The Effects of Dimensionality Curse in High Dimensional kNN Search," in *2011 15th Panhellenic Conference on Informatics*, 41–45.

[21] Pan, S. J., Q. Yang, W. Fan, and S. J. P. (ph. D (2010): "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*.

[22] Roberts, L. G. (1963): "Machine perception of three-dimensional solids," Electrical engineering and computer sciences - ph.d., Massachusetts Institute of Technology.

[23] Rosenblatt, F. (1958): "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," *Psychological Review*, 65–386.

[24] Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li (2014): "ImageNet Large Scale Visual Recognition Challenge," *CoRR*, abs/1409.0575.

[25] S. Mohamed, I. (2017): "Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques," Ph.D. thesis.

[26] Scheirer, W., A. Rocha, R. Micheals, and T. Boult (2010): "Robust Fusion: Extreme Value Theory for Recognition Score Normalization," in *Computer Vision – ECCV 2010*, ed. by K. Daniilidis, P. Maragos, and N. Paragios, Berlin, Heidelberg: Springer Berlin Heidelberg, 481–495.

[27] Scherer, D., A. Müller, and S. Behnke (2010): "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," in *Artificial Neural Networks – ICANN 2010*, ed. by K. Diamantaras, W. Duch, and L. S. Iliadis, Springer Berlin Heidelberg, 92–101.

[28] Simonyan, K. and A. Zisserman (2014): "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv 1409.1556*.

[29] Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014): "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, 15, 1929–1958.

[30] Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2015): "Rethinking the Inception Architecture for Computer Vision," *CoRR*, abs/1512.00567.

[31] Yani, M., M. B. I. S, Si., and M. C. S. S.T. (2019): "Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail," *Journal of Physics: Conference Series*, 1201, 012052.

[32] Y.LeCun (1989): "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, 1(4), 541–551.

[33] Yosinski, J., J. Clune, Y. Bengio, and H. Lipson (2014): "How transferable are features in deep neural networks?" *CoRR*, abs/1411.1792.

A Figures

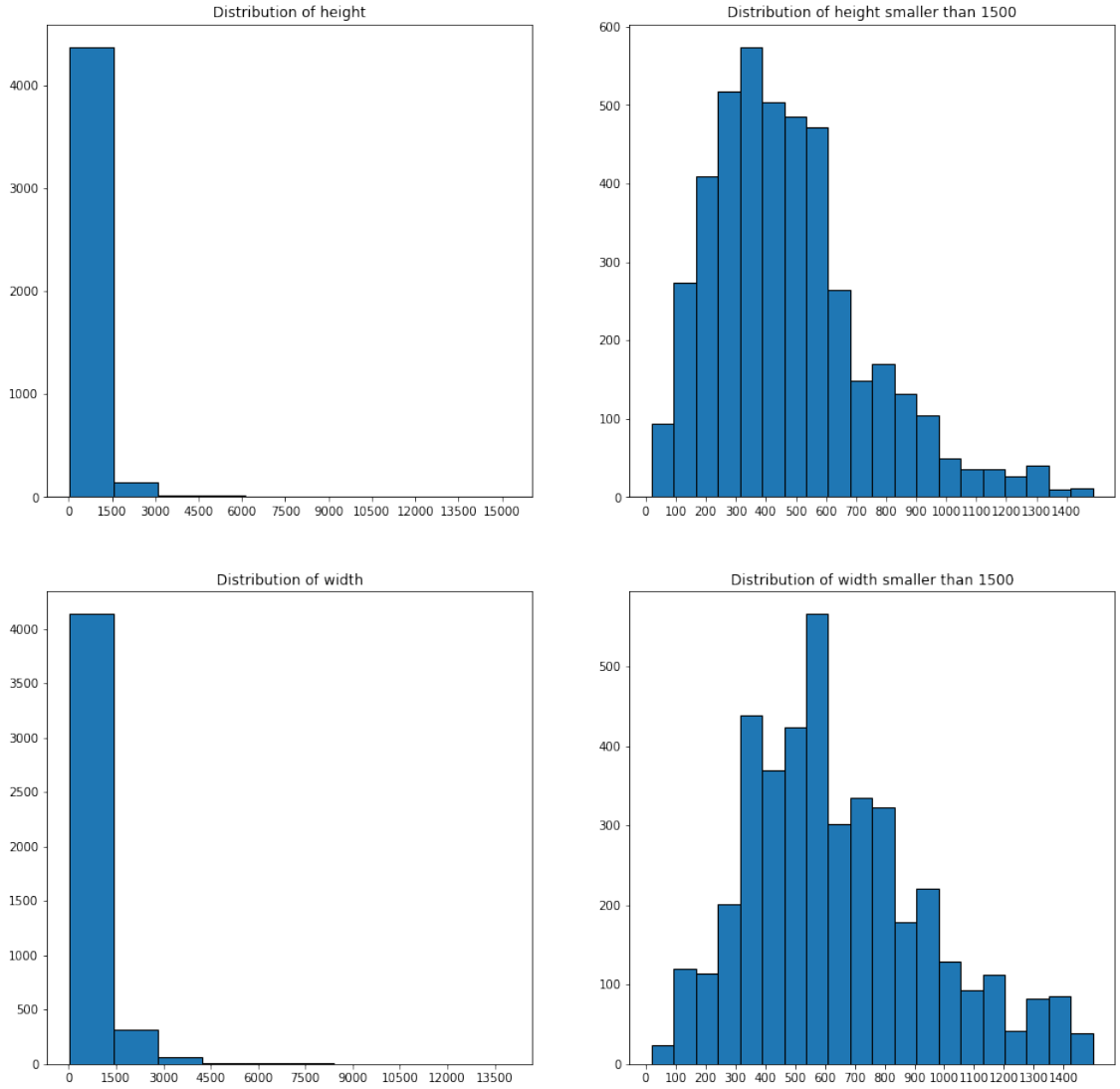


Figure 22: The distribution of image height and width.

B Tables

	Nr. of images
Bar line chart	42
Bar chart	880
Box plots	171
Cartogram	109
Correlogram	8
CFD	52
Dot plot	14
FP	377
Heatmap	12
Histogram	377
Line chart	1493
Pie chart	685
Radar chart	33
Scatterplot	395
Stemplot	14
Violin plot	4

Table 5: Detailed descriptive statistics of number of images for each category in training data.

	Layer Type	Layer Name	Trainable Layer
0	keras.InputLayer object	input_1	False
1	keras.layers.convolutional.Conv2D object	block1_conv1	False
2	keras.layers.convolutional.Conv2D object	block1_conv2	False
3	keras.layers.pooling.MaxPooling2D object	block1_pool	False
4	keras.layers.convolutional.Conv2D object	block2_conv1	False
5	keras.layers.convolutional.Conv2D object	block2_conv2	False
6	keras.layers.pooling.MaxPooling2D object	block2_pool	False
7	keras.layers.convolutional.Conv2D object	block3_conv1	False
8	keras.layers.convolutional.Conv2D object	block3_conv2	False
9	keras.layers.convolutional.Conv2D object	block3_conv3	False
10	keras.layers.pooling.MaxPooling2D object	block3_pool	False
11	keras.layers.convolutional.Conv2D object	block4_conv1	False
12	keras.layers.convolutional.Conv2D object	block4_conv2	False
13	keras.layers.convolutional.Conv2D object	block4_conv3	False
14	keras.layers.pooling.MaxPooling2D object	block4_pool	False
15	keras.layers.convolutional.Conv2D object	block5_conv1	False
16	keras.layers.convolutional.Conv2D object	block5_conv2	False
17	keras.layers.convolutional.Conv2D object	block5_conv3	False
18	keras.layers.pooling.MaxPooling2D object	block5_pool	False
19	keras.layers.core.Flatten object	flatten_1	False
20	keras.layers.core.FC object	dense_1	True
21	keras.layers.core.FC object	dense_2	True
22	keras.layers.core.Prediction object	dense_3	True

Table 6: Trainable layers in VGG-16

	Layer Type	Layer Name	Trainable Layer
0	keras.InputLayer object	input_1	False
1	keras.layers.convolutional.Conv2D object	block1_conv1	False
2	keras.layers.convolutional.Conv2D object	block1_conv2	False
3	keras.layers.pooling.MaxPooling2D object	block1_pool	False
4	keras.layers.convolutional.Conv2D object	block2_conv1	False
5	keras.layers.convolutional.Conv2D object	block2_conv2	False
6	keras.layers.pooling.MaxPooling2D object	block2_pool	False
7	keras.layers.convolutional.Conv2D object	block3_conv1	False
8	keras.layers.convolutional.Conv2D object	block3_conv2	False
9	keras.layers.convolutional.Conv2D object	block3_conv3	False
10	keras.layers.pooling.MaxPooling2D object	block3_pool	False
11	keras.layers.convolutional.Conv2D object	block4_conv1	True
12	keras.layers.convolutional.Conv2D object	block4_conv2	True
13	keras.layers.convolutional.Conv2D object	block4_conv3	True
14	keras.layers.pooling.MaxPooling2D object	block4_pool	True
15	keras.layers.convolutional.Conv2D object	block5_conv1	True
16	keras.layers.convolutional.Conv2D object	block5_conv2	True
17	keras.layers.convolutional.Conv2D object	block5_conv3	True
18	keras.layers.pooling.MaxPooling2D object	block5_pool	True
19	keras.layers.core.Flatten object	flatten_1	True
20	keras.layers.core.FC object	dense_1	True
21	keras.layers.core.FC object	dense_2	True
22	keras.layers.core.Prediction object	dense_3	True

Table 7: Trainable layers of VGG-16 in transfer learning with fine tunning

Declaration of Authorship

I, Sisi Huang hereby confirm that the thesis I am submitting is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Berlin, June 10, 2020

Sisi Huang